

Mapping Toolbox

For Use with MATLAB®

Systems Planning and Analysis, Inc.

Computation

Visualization

Programming

User's Guide

Version 1

How to Contact The MathWorks:



508-647-7000

Phone



508-647-7001

Fax



The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500

Mail



<http://www.mathworks.com>
<ftp.mathworks.com>
<comp.soft-sys.matlab>

Web
Anonymous FTP server
Newsgroup



support@mathworks.com
suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
subscribe@mathworks.com
service@mathworks.com
info@mathworks.com

Technical support
Product enhancement suggestions
Bug reports
Documentation error reports
Subscribing user registration
Order status, license renewals, passcodes
Sales, pricing, and general information

Mapping Toolbox User's Guide

© COPYRIGHT 1997 by The MathWorks, Inc. All Rights Reserved.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

U.S. GOVERNMENT: If Licensee is acquiring the software on behalf of any unit or agency of the U. S. Government, the following shall apply:

(a) for units of the Department of Defense:

RESTRICTED RIGHTS LEGEND: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013.

(b) for any other unit or agency:

NOTICE - Notwithstanding any other lease or license agreement that may pertain to, or accompany the delivery of, the computer software and accompanying documentation, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Clause 52.227-19(c)(2) of the FAR.

Contractor/manufacture is The MathWorks Inc., 24 Prime Park Way, Natick, MA 01760-1500.

MATLAB, Simulink, Handle Graphics, and Real-Time Workshop are registered trademarks and Stateflow and Target Language Compiler are trademarks of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: May 1997 First printing

Preface

Acknowledgments	ii
------------------------------	-----------

Mapping Fundamentals

1

What Is a Map?	1-2
Types of Maps in the Mapping Toolbox	1-3
Vector Maps	1-3
Matrix Maps	1-4
Composite Maps	1-6
Geographic Measurement	1-7
Latitude and Longitude	1-7
Great Circles and Rhumb Lines	1-9
Measuring Distance	1-10
Measuring Azimuth	1-11
Reckoning Positions	1-13
Small Circles	1-13
Measuring Area	1-15
The Geoid Model	1-16
What Is a Geoid?	1-16
The Geoid Vector	1-17
What Is the “Correct” Geoid Vector?	1-18
Measuring the Planets	1-19

Angles, Times, and Distances – Units and Notation	1-21
Angular Notation	1-21
Degrees	1-21
Radians	1-21
Degrees-Minutes-Seconds	1-21
Converting Between Angle Unit Formats	1-22
Time Notation	1-23
Hours	1-23
Seconds	1-23
Hours-Minutes-Seconds	1-23
Converting Between Time Unit Formats	1-24
Distance Notation	1-24
 Working with Vector Maps	 1-25
Data Format	1-25
Segments versus Polygons	1-26
Creating Vector Data	1-27
Calculating Small Circles	1-27
Calculating Tracks – Great Circles and Rhumb Lines	1-28
Geographic Interpolation	1-30
Polygon Area	1-32
Vector Calculations – Intersections	1-34
Trimming Vector Data to a Defined Region	1-36
Simplifying Vector Data	1-38
 Working with Matrix Maps	 1-42
Regular Matrix Maps	1-42
The Map Legend	1-43
The Geographic Meaning	1-44
Accessing Matrix Map Elements	1-47
Valued Maps and Indexed Maps	1-49
Matrix Maps as Logical Variables	1-51
General Matrix Maps	1-53
The Map Format	1-53
Geographic Interpretations	1-56

Introduction to Mapping Graphics	2-2
Map Axes	2-3
Accessing and Manipulating Map Axes Properties	2-6
The Map Frame	2-11
The Map Grid	2-15
Map and Frame Limits	2-18
Switching Between Projections	2-19
Projected and Unprojected Objects	2-22
Displaying Vector Maps as Lines	2-24
Displaying Vector Maps as Patches	2-27
Displaying Matrix Maps	2-30
The Graticule	2-31
Coloring Matrix Maps	2-34
Data Representation	2-36
Image and Surface Coloring	2-36
Surface Light Shading	2-37
Surface Lighted Shaded Relief	2-39
Mapped Light Objects	2-40
The Geographic Data Structure	2-43
Interacting with Displayed Maps	2-47
Specialized Map Displays	2-50
Thematic Map Functions	2-50
Choropleth Maps	2-52

3

Types of Data	3-2
World Vector Data	3-3
Coastlines	3-3
World Atlas Data	3-5
World Matrix Data	3-14
Political	3-14
Terrain	3-17
United States Vector Data	3-18
Low Resolution Data	3-18
Medium Resolution State Outlines	3-23
United States Matrix Data	3-26
Political	3-26
Terrain	3-28
Astronomical Data	3-30

Map Projections

4

What Is a Projection?	4-2
Quantitative Properties	4-3
Geometric Surfaces	4-4
Projection Aspect	4-7
The Origin Vector	4-7
Coordinate Transformations	4-14
Vector Data – rotatem	4-15
Matrix Data – neworig	4-16

Projection Computations	4-18
Summary Guide	4-22

Mapping Applications

5

Introduction to Geostatistics and Navigation	5-2
Geostatistics	5-3
Geographic Means	5-3
Geographic Standard Deviation	5-5
The Meaning of stdm	5-5
The Meaning of stdist	5-6
Equal-Areas in Geostatistics	5-7
Geographic Histograms	5-8
Converting to an Equal-Area Coordinate System	5-10
Geographically Filtering Datasets	5-11
Navigation	5-12
Conventions for Navigational Functions	5-12
Units	5-12
Navigational Track Format	5-13
Fixing Position	5-13
Some Possible Situations	5-14
Using navfix	5-19
A Numerical Example of Using navfix	5-21
Planning	5-27
Track Laydown – Displaying Navigational Tracks	5-28
Dead Reckoning	5-31
Time Zones	5-36
Time Notation	5-38

An Overview	6-2
The Complete GUI Environment: maptool	6-3
Starting maptool	6-3
The Menus	6-4
The Mouse Tool Buttons	6-4
Activating GUIs with maptool: A Working Example	6-5
Setting Initial Map Properties	6-5
Plotting Layered Atlas Data	6-7
Editing Object Properties	6-9
Zooming in on a Map Display	6-11
Entering Workspace Commands	6-12
Plotting Line Objects	6-13
Plotting Text Objects	6-14
Editing Text Position	6-15
Plotting Tracks and Calculating Surface Distances	6-16
Hiding Mapped Objects	6-20
Editing Map Projection and Display Properties	6-22
Selecting Track Waypoints	6-26
Creating a Navigational Track	6-26
Displaying a Navigational Track	6-27
Plotting Patches	6-27
Changing the Stacking Order of Mapped Objects	6-28
Hiding the Mouse Buttons	6-29
Toggling the Map Grid and Frame On	6-30
Constructing Personalized Map Data with GUIs	6-31
Trimming an Existing Data Set	6-31
Encoding a Regular Surface Map	6-33
Creating a Personalized Colormap	6-41

Working With External Data 7-2

Global Vector Data 7-3

 The Digital Chart of the World 7-3

 Looking Up Names and Locations in the DCW Gazetteer .. 7-3

 Importing DCW Themes 7-5

 Reading DCW Files Directly 7-13

U.S. Vector Data 7-14

 TIGER/Line 7-14

 TIGER Thinned Boundary 7-18

 TIGER ArcInfo Format 7-18

 TIGER MapInfo Interchange Format 7-21

Global Gridded Elevation Data 7-23

 ETOPO5 and TerrainBase 7-23

 Digital Chart of the World DEMs 7-28

 U.S. Gridded Elevation Data 7-32

Astronomical Data 7-36

Geographic Terms

Glossary Notes A-2

Glossary A-3

Bibliography



Preface

Acknowledgments

Systems Planning and Analysis, Inc. (SPA) is proud to have developed the Mapping Toolbox. The Mapping Toolbox is the product of the contributions of numerous SPA employees. Ed Byrns and Eric Brown, who are no longer at SPA, conceived the Toolbox, designed the architecture, and have supported the development to its completion. Walter Stumpf managed the completion of the product and also developed the external interface and the atlas data.

Tomi Debole and Andy Kim were essential to the development, testing, and documentation of the Mapping Toolbox. Tom Seaman wrote many of the testing functions. Juliet Crumrine edited the documentation. Wendell Nix helped with the rhumb line calculations and the geometry of track intersections.

The product was developed with long-term support from The MathWorks and many of its key employees, most notably Jim Tung, Scott Gray, Jennifer French, Loren Shure and Clay Thompson.

In memory of John Snyder, whose prior work on Map Projections inspired and made possible the creation of the Mapping Toolbox.

Mapping Fundamentals

What Is a Map	1-2
Types of Maps in the Mapping Toolbox	1-3
Vector Maps	1-3
Matrix Maps	1-4
Composite Maps	1-6
Geographic Measurement	1-7
Latitude and Longitude	1-7
Great Circles and Rhumb Lines	1-9
Measuring Distance	1-10
Measuring Azimuth	1-11
Reckoning Positions	1-13
Small Circles	1-13
Measuring Area	1-15
The Geoid Model	1-16
Measuring the Planets	1-19
Angles, Times, and Distances – Units and Notation	1-21
Angular Notation	1-21
Time Notation	1-23
Distance Notation	1-24
Working with Vector Maps	1-25
Data Format	1-25
Segments versus Polygons	1-26
Creating Vector Data	1-27
Geographic Interpolation	1-30
Polygon Area	1-32
Vector Calculations – Intersections	1-34
Trimming Vector Data to a Defined Region	1-36
Simplifying Vector Data	1-38
Working with Matrix Maps	1-42
Regular Matrix Maps	1-42
General Matrix Maps	1-53

What Is a Map?

The simplest definition of a *map* is “a representation of geographic data.” To the ancient Egyptians, this representation first took the form of lists of city names in the order they would be encountered when following a given road. Two-dimensional renditions, such as paper road maps, are more familiar to us today. Even classroom globes are maps under this definition.

This toolbox addresses electronic representations of geographic data. It allows the creation, use, and presentation of geographic data in a variety of forms and to a variety of ends.

In the Mapping Toolbox, a *map* is any variable or set of variables representing or assigning values to a geographic location or region, from a single point to an entire planet.

Types of Maps in the Mapping Toolbox

Vector Maps

A series of latitude-longitude coordinate pairs representing, for example, points along the coast of Greenland, or along Interstate 80, or even the two sets together, form a map. In this case, the geographic data is in *vector* format and is referred to as a *vector map*. This format consists of specific points, along with some indication as to how they should or should not be connected to each other. In the Mapping Toolbox, vector data consists of sequentially ordered pairs of latitude and longitude coordinates. The pairs are considered to be connected in sequence; breaks in connectivity must be delineated by the creation of separate variables or by inserting NaNs into the sets at the appropriate break points. For vector map data, the connectivity of the data is often only a concern during display.

To illustrate a vector map, enter the following:

```
load coast
whos
```

Name	Size	Bytes	Class
lat	9589x1	76712	double array
long	9589x1	76712	double array

```
axesm mercator
framem
plotm(lat, long)
```

The variables `lat` and `long` are vectors in the `coast` MAT-file, which together form a vector map of the coastlines of the world. They form a map independent of whether they are displayed.

We have chosen to view the map in a Mercator projection:



Matrix Maps

A matrix in which each element represents a value corresponding to a specific geographic area also forms a map. In this case, the geographic data is in *matrix* format, which is often called *raster* format. However, the Mapping Toolbox makes great use of the powerful matrix manipulation capabilities of MATLAB to fully exploit this map type, so the term *matrix map* is more appropriate.

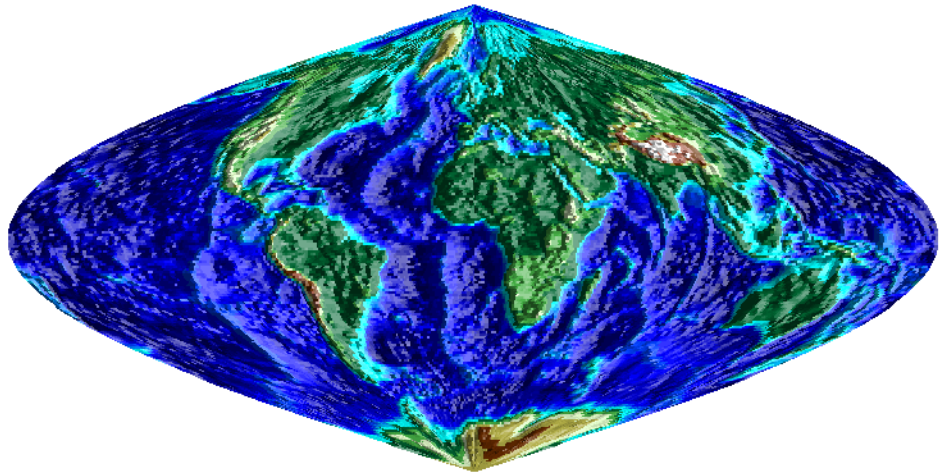
When the data consists of surface elevations, the map can also be referred to as a *digital elevation map* (DEM) or *topographical map*.

As an example of this map type, consider a 180-by-360 matrix. Each row represents one degree of latitude, and each column represents one degree of longitude. Each entry of this matrix is the average elevation, in meters, for the one-degree-by-one-degree region of the Earth to which its row and column correspond. This is, in fact, the `topo` matrix that is provided in the MATLAB product in the workspace of the same name.

To display the matrix map, type the following:

```
clm    % clears map  
load topo  
axesm sinusoid  
meshl srm(topo, topol legend)
```

The `topo` matrix is displayed as a lighted, shaded relief map in a Sinusoidal projection:



Note that the matrix logic of the map and the projection in which it is displayed are completely independent. The `topo` variable is a map whether it is displayed or not, and it is the same map no matter how it is displayed.

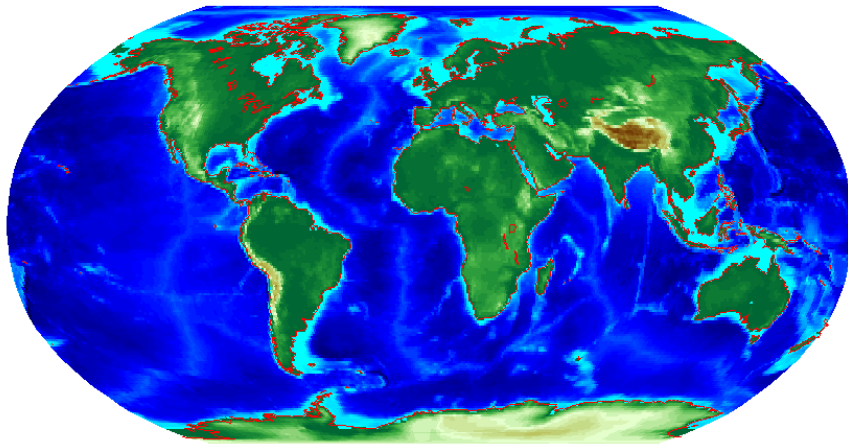
Composite Maps

Many occasions arise in which vector map variables and matrix map variables are used or displayed together. For example, continental coastlines in vector form might be displayed with matrix temperature data to make the latter more useful. When several map variables are used together, regardless of type, they can be correctly referred to as a single map.

Since we have our `coast` and `topo` workspaces available from the previous examples, we can combine the two in a single map and see how well the two types of data correspond:

```
figure; axesm robinson  
meshm(topo, topol legend); demcmap(topo)  
plotm(lat, long, 'r')
```

Here is the resulting map:



Note that the map has been changed from a Sinusoidal to a Robinson projection. Essentially, we have displayed our matrix map of elevations and dropped the coastal vector data on top of it. The red coastal lines fall quite nicely around the edges of the continents.

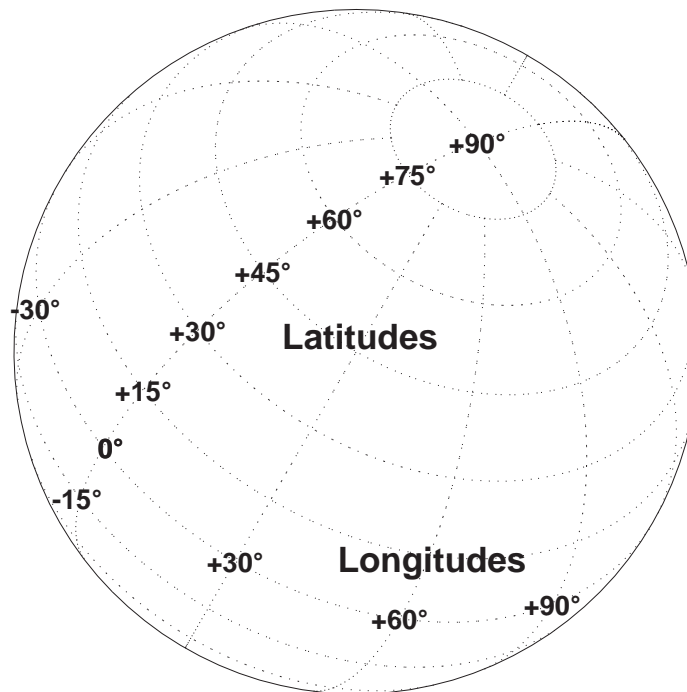
The remainder of this chapter will focus on the fundamental principles of geographic measurement and data manipulation that are a prerequisite for creating map displays. Chapter 2, “Displaying Maps”, takes up the topic of map display and interaction.

Geographic Measurement

Latitude and Longitude

The position of a point on the surface of the Earth, or any other planet, for that matter, can be specified with two angles, *latitude* and *longitude*. These angles can be specified in degrees or radians; however, degrees are far more common in geographic usage.

Latitude is the angle at the center of the planet between the plane of the Equator and a line through the center passing through the surface at the point in question. Latitude is positive in the Northern Hemisphere, reaching a limit of $+90^\circ$ at the North Pole, and negative in the Southern Hemisphere, reaching a limit of -90° at the South Pole. Lines of constant latitude are called *parallels*.



Longitude is the angle at the center of the planet between two planes passing through the center and perpendicular to the plane of the Equator. The first plane also includes the surface point in question; the second plane contains the Prime Meridian, which currently means it includes the point defined by the Royal Observatory in Greenwich, England. Lines of constant longitude are called *meridians*.

Longitudes ranging from -180° to $+180^\circ$ are unique. Longitudes may be given outside this range, but such values can always be renamed to a value within the range. The Mapping Toolbox provides a number of functions to convert data between different latitude and longitude ranges. For example, the command `npi 2pi` wraps data to this domain:

```
longi tudes = [-560 125 190];
newl ongi tudes = npi 2pi (l ongi tudes)
newl ongi tudes =
    160.0000    125.0000   -170.0000
```

Sometimes it is more useful to consider longitude as strictly positive, proceeding from the Prime Meridian (0°) eastward around and back to the Prime Meridian (360°). Any longitude data can be converted to this domain using the `zero22pi` command:

```
posi ti vel ongs = zero22pi (newl ongi tudes)
posi ti vel ongs =
    160.0000    125.0000    190.0000
```

If you need this data in radians, you can use an angle conversion function:

```
radi anl ongs = deg2rad(posi ti vel ongs)
radi anl ongs =
    2.7925    2.1817    3.3161
```

Several angle conversion functions are available in this toolbox, supporting degrees, radians, and degrees-minutes-seconds notation.

What is the antipodal point (on the opposite side of the Earth) of Natick, Massachusetts (about 42.3°N, 71.35°W)?

```
[ antilat, antilong] = antipode(42. 3, -71. 35)
antilat =
    -42. 3000
antilong =
    108. 6500
```

The result (42.3°S,108.65°E) lies in the Indian Ocean southwest of Australia.

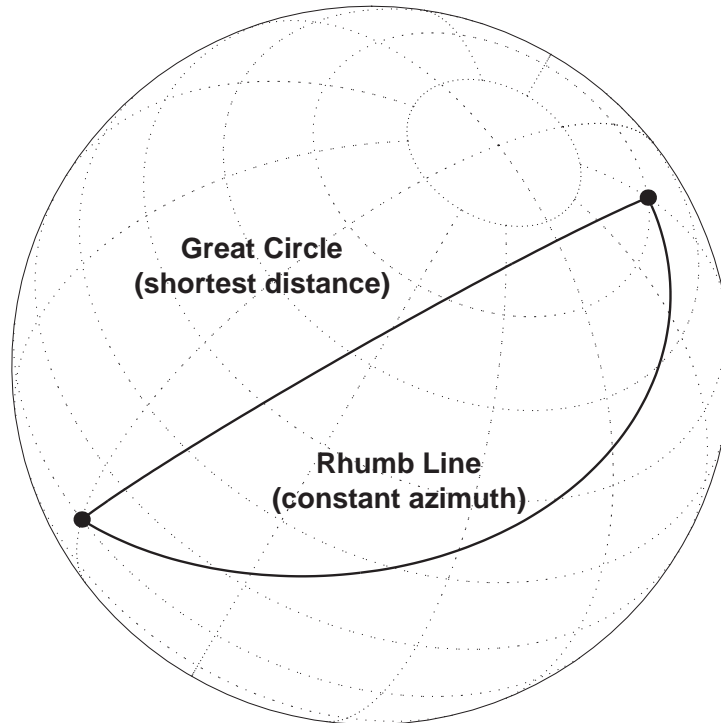
Great Circles and Rhumb Lines

In plane geometry, lines have two important characteristics. A line represents the shortest path between two points, and the slope of such a line is constant. When describing lines on the surface of a spheroid, however, only one of these characteristics may be guaranteed at a time.

In geography, a *great circle* is the shortest path between two points along the surface of a sphere. The precise definition of a great circle is the intersection of the surface with a plane passing through the center of the planet. In general, great circles do not have a constant azimuth, the spherical analog of slope; they cross successive meridians at different angles. When the Earth is taken as a sphere, the Equator and all meridians are great circles. Great circles always bisect the sphere.

A *rhumb line* is a complex curve, the special property of which is that it crosses each meridian at the same angle. This curve is also referred to as a *loxodrome*. In general, a rhumb line is not the shortest path between two points on the rhumb line. All parallels, including the Equator, are rhumb lines, since they cross all meridians at 90°. Additionally, all meridians are rhumb lines. Rhumb lines always terminate at the poles, unless the azimuth is true east or west, in which case the rhumb line closes on itself to form a parallel of latitude.

A description of how to calculate points along great circles and rhumb lines appears later in this document.



Measuring Distance

When you calculate the distance between two points with the Mapping Toolbox, the result will depend upon whether you want a great circle or a rhumb line distance. The `distance` command will return the appropriate distance between two points as an angular arc length, employing the same angular units as the input latitudes and longitudes. The default path type is the shorter great circle, and the default angular units are degrees. The previous figure shows two points at (15°S, 0°) and (60°N, 150°E). The great circle distance between them, in degrees of arc, is as follows:

```
distgc = distance(-15, 0, 60, 150)
distgc =
    129.9712
```

The rhumb line distance is greater:

```
distrh = distance(' rh' , -15, 0, 60, 150)
distrh =
    145.0288
```

To determine how much longer the rhumb line path is in, say, kilometers, you can use a distance conversion function on the difference:

```
kmdifference = deg2km(distrh-distgc)
kmdifference =
    1.6744e+03
```

Several distance conversion functions are available in the toolbox, supporting degrees, radians, kilometers, statute miles, and nautical miles. Converting distances between angular arc length units and surface length units requires the radius of a planet or spheroid. By default, the radius of the Earth is used.

Measuring Azimuth

Azimuth is the angle a line makes with a meridian, taken clockwise from north. When the azimuth is calculated from one point to another using the Mapping Toolbox, the result will depend upon whether a great circle or a rhumb line azimuth is desired. For great circles, the result will be the azimuth at the starting point of the connecting great circle path. In general, the azimuth along a great circle is not constant. For rhumb lines, the resulting azimuth is constant along the entire path.

Azimuths, or bearings, are returned in the same angular units as the input latitudes and longitudes. The default path type is the shorter great circle, and the default angular units are degrees. In our example, the great circle azimuth from the first point to the second is:

```
azgc = azimuth(-15, 0, 60, 150)
azgc =
    19.0391
```

For the rhumb line, the constant azimuth is:

```
azrh = azimuth(' rh' , -15, 0, 60, 150)
azrh =
    58.8595
```

One feature of rhumb lines is that the inverse azimuth, from the second point to the first, is the complement of the forward azimuth and can be calculated by simply adding 180° to the forward value:

```
i n v e r s e r h = a z i m u t h ( ' r h ' , 60, 150, -15, 0)
i n v e r s e r h =
238.8595

d i f f e r e n c e = i n v e r s e r h - a z r h
d i f f e r e n c e =
180
```

This is not true, in general, of great circles:

```
i n v e r s e g c = a z i m u t h ( ' g c ' , 60, 150, -15, 0)
i n v e r s e g c =
320.9353

d i f f e r e n c e = i n v e r s e g c - a z g c
d i f f e r e n c e =
301.8962
```

The azimuths associated with cardinal and intercardinal compass directions are the following:

North	0° or 360°
Northeast	45°
East	90°
Southeast	135°
South	180°
Southwest	225°
West	270°
Northwest	315°

Reckoning Positions

A common problem in geographic applications is the determination of a destination given a starting point, an initial azimuth, and a distance. In the Mapping Toolbox, this process is called *reckoning*. A new position may be reckoned in a great circle or a rhumb line sense (great circle or rhumb line track).

As an example, an airplane takes off from La Guardia Airport in New York (40.75°N, 73.9°W) and follows a northwestern rhumb line flight path at 200 knots (nautical miles per hour). Where would it be after 1 hour?

```
[rhl at, rhl ong] = reckon(' rh' , 40. 75, -73. 9, nm2deg(200) , 315)
rhl at =
    43. 1054
rhl ong =
   -77. 0665
```

Notice that the distance, 200 nautical miles, must be converted to degrees of arc length with the `nm2deg` conversion function to match the latitude and longitude inputs. If the airplane had a flight computer that allowed it to follow an exact great circle path, what would the aircraft's new location be?

```
[gcl at, gcl ong] = reckon(' gc' , 40. 75, -73. 9, nm2deg(200) , 315)
gcl at =
    43. 0615
gcl ong =
   -77. 1238
```

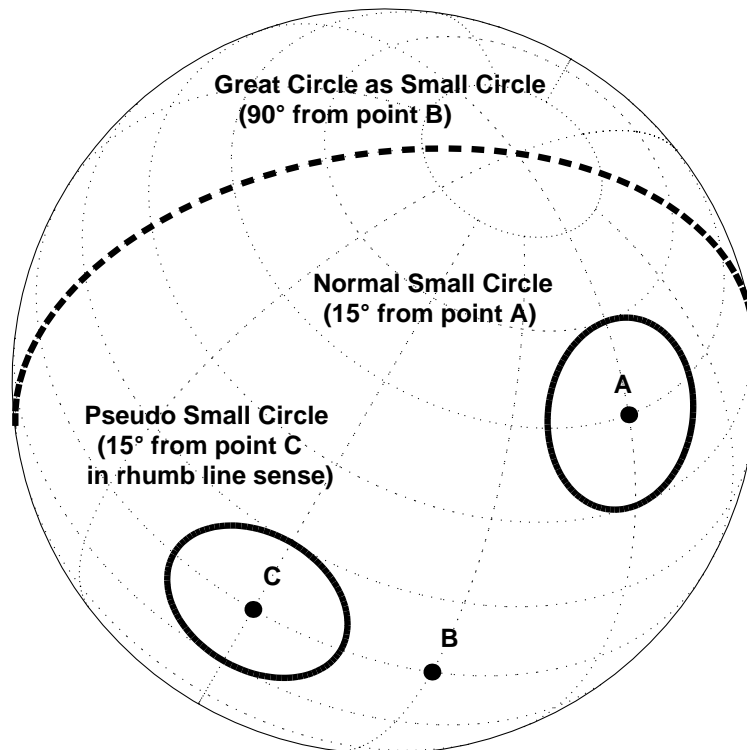
Notice also that for short distances at these latitudes, the result hardly differs between great circle and rhumb line. The two destination points are less than 4 nautical miles apart. Incidentally, after 1 hour, the airplane would be just north of New York's Finger Lakes.

Small Circles

In addition to rhumb lines and great circles, one other geographic construction is significant in geography and the Mapping Toolbox: the *small circle*. The precise definition of a small circle is the intersection of a plane with the surface of the planet. In the Mapping Toolbox, this definition includes planes passing through the center of the planet, so the set of all small circles includes all great circles as limiting cases. This usage is not universal.

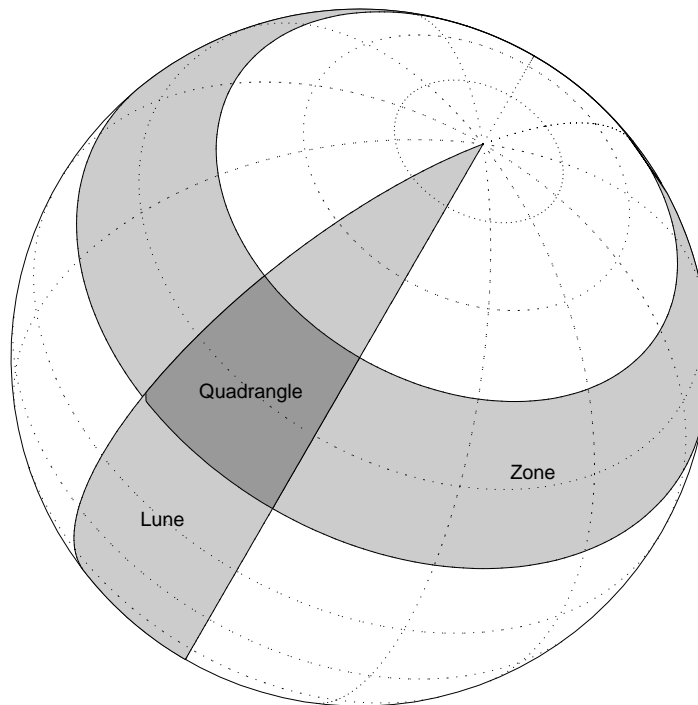
Small circles are most easily defined by distance. “All points 45 nm (nautical miles) distant from (45°N,60°E)” would be the description of one small circle. If degrees of arc length are used as a distance measurement, then a great circle is the set of all points 90° distant from a particular “center” point.

For true small circles, the distance must be defined in a great circle sense, the shortest distance between two points on the surface of a sphere. However, the Mapping Toolbox also allows the calculation of a “pseudo small circle,” for which distances are measured in a rhumb line sense (along lines of constant azimuth). These should not be confused with true small circles.



Measuring Area

In solid geometry, the area of a spherical quadrangle can be exactly calculated. A spherical quadrangle is the intersection of a *lune* and a *zone*. In geographic terms, a *quadrangle* is defined as a region bounded by parallels north and south, and meridians east and west.



In the pictured example, a quadrangle is formed by the intersection of a zone, which is the region bounded by 15°N and 45°N latitudes, and a lune, which is the region bounded by 0° and 30°E longitude. Under the spherical planet assumption, the fraction of the entire spherical surface area inscribed in the quadrangle can be calculated:

```
area = areaquad(15, 0, 45, 30)
area =
    0.0187
```

That is, less than 2% of the planet's surface area is in this quadrangle. To get an absolute figure in, for example, square miles, you must provide the appropriate spherical radius. The radius of the Earth is about 3958.9 miles.

```
area = areaquad(15, 0, 45, 30, 3958.9)
area =
    3.6788e+06
```

The surface area within this quadrangle is over 3.6 million square miles for a spherical Earth.

The Geoid Model

Although the Earth is round, it is not exactly a sphere. However, for many users and most applications, the difference is so small that a spherical assumption is sufficient. This section addresses how the Mapping Toolbox handles more accurate models of the shape, or figure, of the Earth and other planets.

What Is a Geoid?

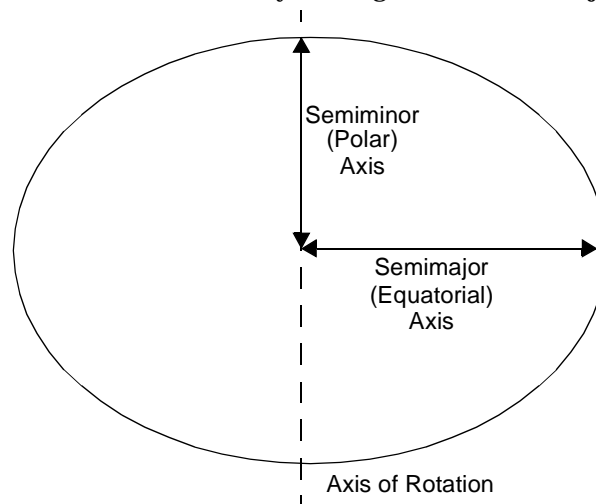
Literally, “geoid” means “Earth-shaped.” The *geoid* is the precise figure of the Earth. Specifically, it is an equipotential surface with respect to gravity. It is approximately an oblate ellipsoid, but not exactly so because of local variations in gravity.

For the purposes of the Mapping Toolbox, the geoid of the Earth is assumed to be an exact ellipsoid. This is a simplification of the precise definition of geoid; however, it is a good approximation, which allows a great variety of calculations to be performed that would otherwise be prohibitively complicated. Ellipsoidal models of the Earth's figure are very common in practice and are considered sufficient for cartography. The use of the term geoid for this ellipsoid is perhaps peculiar to the toolbox. Further, in the Mapping Toolbox, the term geoid is extended to ellipsoidal models for the figures of the Sun, Moon, and planets.

The Geoid Vector

Since the toolbox treats the geoid as an ellipsoid, the geoid can be described with a two-element vector, which is often called the *geoid vector* in this guide. The geoid vector has the form `[semi major-axis eccentricity]`. The semimajor axis can be in any unit of distance; the choice of these units often drives the units used for distance outputs in the toolbox functions.

An *ellipsoid* is defined as a rotation of an ellipse around its minor axis. Ellipses can be defined with a semimajor and a semiminor axis, “flattening,” the parameter “n,” or the preferred method of a semimajor axis and eccentricity. The toolbox has functions to convert elliptical definitions from any of these forms to the form of the geoid vector. For example, the command `axes2ecc` returns an eccentricity when given the semimajor and semiminor axes.



The geoid vector, along with several other kinds of planetary data for each of the planets and the Earth’s moon, can be queried using the `almanac` function provided by the Mapping Toolbox (see “Measuring the Planets”).

For the Earth, the semimajor axis is about 21 kilometers longer than the semiminor axis. Use the `almanac` function to verify this:

```
geoid = almanac('earth', 'geoid', 'kilometers')
geoid =
    6378.14    0.08

semiminor = minaxis(geoid)
semiminor =
    6356.75

geoid(1) - semiminor
ans = 21.38
```

When compared to the semimajor axis, which is almost 6400 kilometers, this difference seems insignificant and can be neglected for most purposes. For this reason, most functions in the Mapping Toolbox default to a spherical model of the Earth. The ellipse in the previous diagram is very exaggerated. The difference between the semimajor and semiminor axes of a correct ellipse at this scale would be far smaller than the width of the line used to draw it.

What Is the “Correct” Geoid Vector?

A variety of ellipsoid models have been proposed through the years. They differ because of the surveying information upon which they are based, or because certain regions of interest of the true geoid are best modeled by a particular ellipsoid that may be less accurate for other parts of the world. The Mapping Toolbox default geoid vector (after the sphere) is based on the 1980 Geodetic Reference System ellipsoid. This geoid vector is returned by the statement `almanac('earth', 'geoid')`. It is also the reference ellipsoid for the 1984 World Geodetic System equipotential model.

Several other geoid vectors are supported, for models ranging from Everest's 1830 ellipsoid (used for India) to the International Astronomical Union ellipsoid of 1965 (used for Australia). These can be referenced by the following statements:

```
geoid1 = almanac('earth', 'geoid', [], 'everest');
geoid2 = almanac('earth', 'geoid', [], 'iau65');
```

The reference page for the `almanac` function has more information on which geoids are available.

You should probably use a spherical Earth unless you have a good reason to use an ellipsoidal model. If you use an ellipsoidal model, you should probably use the default unless you need a specific one. If you cannot find the geoid vector you need, you can create it in the following form:

```
geoidvec = [semi major-axis eccentricity]
```

Note that the default units for the geoid semimajor axis in the `almanac` function are kilometers, which can be used by simply passing in an empty matrix in place of the input units string, as in the previous example. Eccentricity is dimensionless.

Measuring the Planets

The Mapping Toolbox contains a function that provides almanac data on the major bodies of our solar system. Geophysical data, such as geoid vectors, radii, surface areas, and volumes, can be accessed for the Sun, the Earth's moon, and all of the planets, in any of the supported units of distance measurement.

Many planets have ellipsoidal geoid vectors available. Some planets return spherical geoid vectors only:

```
almanac('earth', 'geoid', 'nautical miles')
ans =
    3443.92         0.08
```

```
almanac('mars', 'geoid', 'kilometers')
ans =
    3396.90         0.11
```

```
almanac('moon', 'geoid', 'statutemiles')
ans =
    1079.97         0
```

When a radius is desired, a scalar is returned representing the radius of the best spherical model of the planet. Notice that for a spherical model, the radius in radians is 1:

```
almanac('mercury','radius','kilometers')
ans =
    2439

almanac('neptune','radius','radians')
ans =
    1
```

Surface areas and volumes are calculated based on a spherical model by default. In most cases, you can use the geoid model instead, and for the Earth, you can specify any of the supported geoid models. You can also request the actual tabulated values of the Earth:

```
almanac('mars','surfacearea','kilometers','geoid')
ans =
    1.4441e+08

almanac('earth','volume','kilometers','international')
ans =
    1.0833e+12

almanac('earth','volume','kilometers','actual')
ans =
    1.0832e+12
```

A complete description of available data is provided under the `almanac` reference page in the “Mapping Reference” chapter of the *Mapping Toolbox Reference Guide*.

Angles, Times, and Distances – Units and Notation

Angular Notation

Angles can be represented as variables in the Mapping Toolbox in three ways: degrees, radians, and degrees-minutes-seconds. The toolbox provides functions for converting between these formats.

Degrees

This is the default angular unit notation for the toolbox. Although for most scientific applications, radians are more commonly used, in geographic usage, degrees are much more convenient.

Degree notation is simply decimal notation in terms of degrees. Forty-three and one-half degrees would be 43.5.

Radians

Radian notation is simply decimal notation in terms of radians. Two radians would be 2.0.

Degrees-Minutes-Seconds

Degrees-minutes-seconds, or “dms,” notation is a little more complicated. In text, a “dms” angle would be *ddd° mm' ss"*. For example, 142°15'27" is 142 degrees, 15 minutes, and 27 seconds. There are 60 seconds in a minute and 60 minutes in a degree. In the Mapping Toolbox, when “dms” angles are represented by a single number, the format is *ddmm.ss*. For example, 142°15'27" is 14215.27.

The real value of this notation is for the input of data already in this format. The toolbox includes the `mat2dms` function for easily entering “dms” data.

A special case of the “dms” format is the “dm” format, in which seconds are not included.

If you have a three-column matrix in which the columns are degrees, minutes, and seconds, respectively, `mat2dms` will convert it to “dms” format:

```
dmsmatrix = [45 13 46; 156 45 01; -7 34 12.1]
dmsmatrix =
    45.0000    13.0000    46.0000
   156.0000    45.0000     1.0000
    -7.0000    34.0000    12.1000

dmsformat = mat2dms(dmsmatrix)
dmsformat =
    1.0e+04 *
    0.4513
    1.5645
   -0.0734
```

NOTE: Care must be exercised when working with the “dms” format; for example, two angles in this format cannot simply be added. It is advisable to convert “dms” data to decimal degrees before working extensively with it.

Converting Between Angle Unit Formats

The toolbox includes a variety of angle unit conversion functions. For example, to convert the “dms” format values to degrees or to radians, you can use `dms2deg` and `dms2rad`, respectively:

```
degformat = dms2deg(dmsformat)
degformat =
    45.2294
   156.7503
   -7.5700

radformat = dms2rad(dmsformat)
radformat =
    0.7894
    2.7358
   -0.1321
```

Similar functions include `deg2rad`, `rad2deg`, `deg2dms`, etc. There is also a more general function, `anglem`, which will convert from one format to another. For example, how many degrees are in $\frac{1}{4}$ radians?

```
degs = anglem(1/4*pi, 'radians', 'degrees')
degs =
    45
```

Time Notation

Times can be represented as variables in the Mapping Toolbox in three ways: hours, seconds, and hours-minutes-seconds. The toolbox provides functions for converting between these formats.

Hours

This is the default time unit notation for the toolbox.

Hour notation is simply decimal notation in terms of hours. Two hours and fifteen minutes would be 2.25.

Seconds

Seconds notation is simply decimal notation in terms of seconds. One hour would be 3600.

Hours-Minutes-Seconds

Hours-minutes-seconds, or “hms” notation, is analogous to “dms” notation for angles. In text, an “hms” time would be *hh:mm:ss*. For example, 12:36:15 is 12 hours, 36 minutes, and 15 seconds. In the Mapping Toolbox, when “hms” times are represented by a single number, the format is *hhmm.ss*. For example, 12:36:15 is 1236.15.

The real value of this notation is in entering data already in this format. The toolbox includes the `mat2hms` function to easily input “hms” data, which functions very similarly to the `mat2dms` function described earlier.

NOTE: Care must be exercised when working with the “hms” format; for example, two times in this format cannot simply be added. It is advisable to convert “hms” data to decimal hours before working extensively with it.

Converting Between Time Unit Formats

Time units can be converted using functions similar to those described for angle unit conversions. These include `hr2sec` and `hms2hr`, as well as a general conversion function `timedim`, which works just like `angl edim`.

Distance Notation

Distances in the Mapping Toolbox can be measured in a number of different units. Functions are provided to convert between nautical miles (nm), statute miles (sm), kilometers (km), degrees of arc length (deg), and radians of arc length (rad). The names of these functions are of the form `sm2km`, `km2rad`, etc. A general distance conversion function, `distdim`, is available as well.

There is no single default unit of distance measurement in the toolbox. Navigation functions use nautical miles as a default, almanac functions default to kilometers, and the `distance` function defaults to degrees of arc length. It is essential that you understand the default units of any function you use.

NOTE: When distances are given in terms of angular units (degrees or radians), be careful to remember that these are in terms of arc length. While a degree of latitude always subtends a degree of arc length, this is not necessarily the case for degrees of longitude. Also, using arc length as distance assumes a knowledge of the underlying radius.

On the Earth, a degree of arc length is about 60 nautical miles:

```
nautical miles = deg2nm(1)
nautical miles =
    60.0405
```

The Earth is the default assumption for these conversion functions. Other radii may be used, however:

```
nautical miles = deg2nm(1, almanac('moon', 'radius'))
nautical miles =
    30.3338
```


Working with Vector Maps

Data Format

Vector maps are usually contained in a pair of variables, which represent the latitude-longitude pairs for points of interest. For example, the following two variables can be a vector map:

```
lat = [45.6 -23.47 78];
long = [13 -97.45 165];
```

Since this is a made-up example, these points could mean anything. Perhaps they are the locations over which geosynchronous satellites are stationed. If so, you might think of them as three distinct points. Perhaps they represent the starting point, the mid-course marker, and the finish point of an airplane race. Think of them as three points connected by two line segments. Or perhaps they represent the vertices of a triangle bounding a region with a propensity for unexplained phenomena, in which case think of them as describing a polygon.

The Mapping Toolbox provides functionality for each of these interpretations. For many purposes, the distinction is irrelevant; for others, the choice of a function implies the appropriate interpretation. For example, the command `plotm` will display the data as a line, while `fillm` will display it as a filled polygon.

When vector variables are used to represent several segments or polygons, successive objects should be delineated by the insertion of NaNs in both variables. For example, if a second segment is to be added to the above map, the two objects can reside in the same pair of variables:

```
lat = [45.6 -23.47 78 NaN 43.9 -67.14 90 -89];
long = [13 -97.45 165 NaN 0 -114.2 -18 0];
```

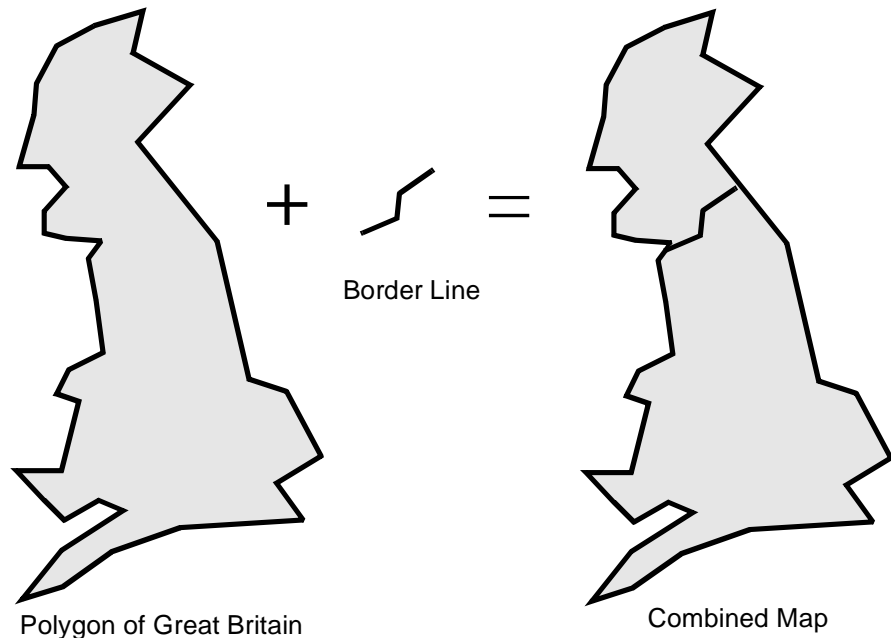
Notice that the NaNs must appear in the same locations in both variables. Here we have a segment of three points separated from a segment of four points. The NaNs perform two functions; they provide a simple means of identifying break points in the data, and they act as “pen-up” commands when plotting vector maps. The NaNs are also used in the Mapping Toolbox to separate non-connected patch faces.

NOTE: This definition differs from MATLAB, which does not display NaN clipped data as patches.

Segments versus Polygons

Geographic objects represented by vector data may or may not be formatted as polygons. Imagine two variables, *l at coast*, and *l oncoast*, which correspond to sequential points around the coast of the island of Great Britain. If this data returns to its starting point, then a polygon-formatted map of Great Britain exists. This data might be plotted as a patch or as a line, and it might be logically employed in calculations as either.

Now consider two more variables, *l at border* and *l onborder*, which correspond to points along the Anglo-Scottish border, proceeding from the west coast at Solway Firth to the east coast at Berwick-upon-Tweed. This data can really only be plotted as a line. The two pairs of variables together can form a map. They can be displayed as a patch of Great Britain with a line showing the border. However, you still would not have a polygon-formatted map of Scotland alone.



Creating Vector Data

In addition to the vector map data available in the Mapping Toolbox and from external sources for such things as coastlines, political boundaries, and other more “permanent” objects, you may want to create your own vector data for such things as great circles, rhumb lines, and small circles.

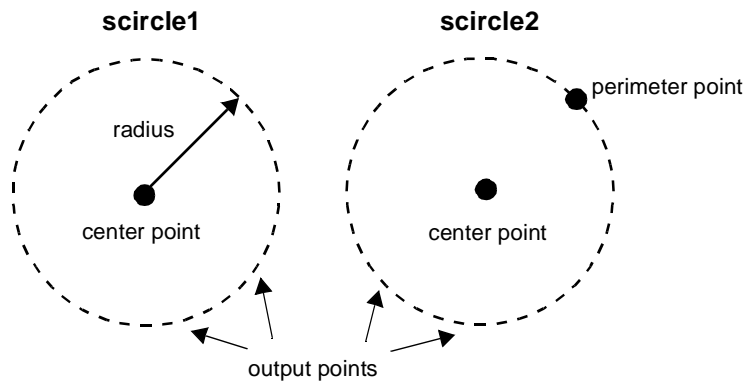
Calculating Small Circles

You can calculate vector data for points along a small circle in two ways. If you have a center point and a known radius, use `scircle1`; if you have a center point and a single point along the circumference of the small circle, use `scircle2`. For example, to get data points describing the small circle at 10° distance from (67°N, 135°W), use the following:

```
[latc, lonc] = scircle1(67, -135, 10);
```

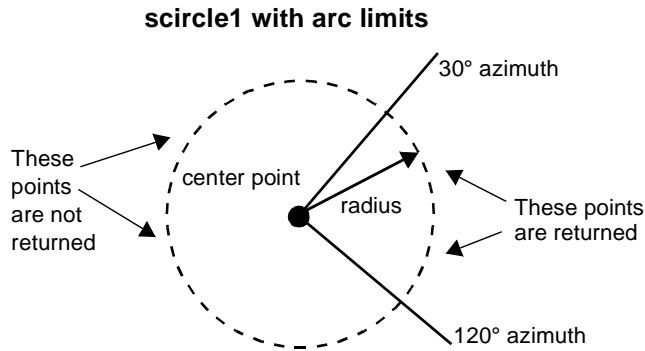
To get the small circle centered at the same point that passes through the point (55°N, 135°W), use `scircle2`:

```
[latc, lonc] = scircle2(67, -135, 55, -135);
```



The `scircle1` function also allows points along a specific arc of the small circle to be calculated. For example, if you want to know the points 10° in distance and between 30° and 120° in azimuth from (67°N,135°W), simply provide arc limits:

```
[latc, longc] = scircle1(67, -154, 10, [30 120]);
```



When an entire small circle is calculated, the data is in polygon format. For all calculated small circles, 100 points are returned unless otherwise specified. You can calculate several small circles at once by providing vector inputs. See the reference pages on `scircle1` and `scircle2` in the *Mapping Toolbox Reference Guide* for more information.

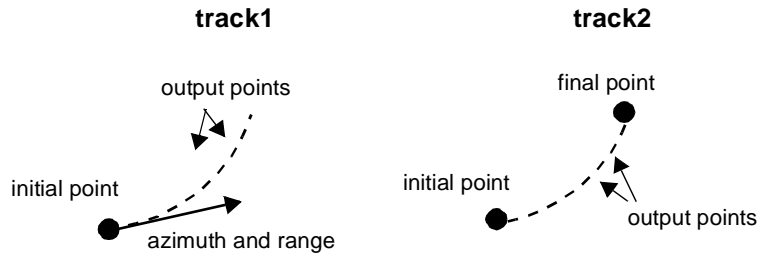
Calculating Tracks – Great Circles and Rhumb Lines

You can generate vector data corresponding to points along great circle or rhumb line tracks using `track1` and `track2`. If you have a point on the track and an azimuth at that point, use `track1`. If you have two points on the track, use `track2`. For example, to get the great circle path starting at (31°S, 90°E) with an azimuth of 45° with a length of 12°, use `track1`:

```
[latgc, longc] = track1('gc', -31, 90, 45, 12);
```

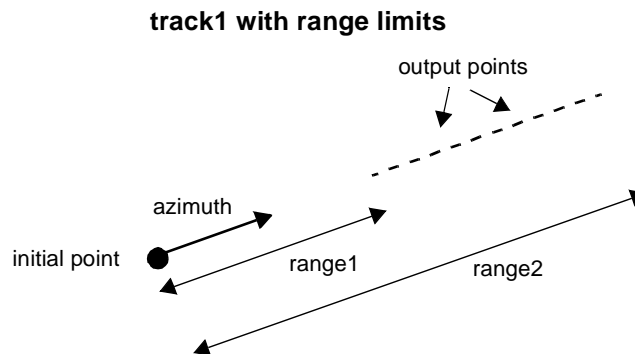
For the great circle from (31°S, 90°E) to (23°S, 110°E), use track2:

```
[latgc, longc] = track2('gc', -31, 90, -23, 110);
```



The track1 function also allows range endpoints to be specified. For example, if you want points along a rhumb line starting 5° away from the initial point and ending 13° away, at an azimuth of 55°, simply specify the range limits:

```
[latrh, lonrh] = track1('rh', -31, 90, 55, [5 13]);
```



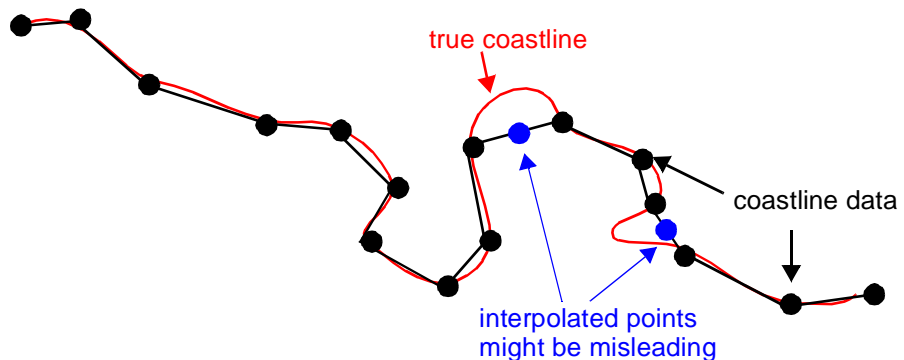
When no range is provided for track1, the returned points represent a “complete track.” For great circles, a complete track is 360°, encircling the planet and returning to the initial point. For rhumb lines, the complete track terminates at the poles, unless the azimuth is 90° or 270°, in which case the complete track is a parallel that returns to the initial point.

For calculated tracks, 100 points are returned unless otherwise specified. You can calculate several tracks at one time by providing vector inputs. See the reference pages on `track1` and `track2` in the *Mapping Toolbox Reference Guide* for more information. More vector path calculations are described later in Chapter 5 of this document under the section “Navigation.”

Geographic Interpolation

When using vector data, you must be careful when you make assumptions concerning geographic reality between data points. For instance, when plotting vector data, you might connect each point with a straight line segment. This does not usually indicate any true knowledge about the region between known points. Data consisting of points along a coastline might be sparse; in the absence of other knowledge, filling in data can be misleading.

Interpreting Sparse Vector Data



Despite the dangers of misinterpretation, many circumstances exist in which geographic data interpolation is useful or necessary. Sparser data can be linearly “filled-in” with the `interp` function.

Consider a set of latitude and longitude points that you want to be no further than one degree in separation in either direction.

```

lats = [1 2 4 5]; longs = [1 3 4 5]; maxdiff = 1;

[newlats, newlongs] = interp(lats, longs, maxdiff)
newlats =
    1.0000
    1.5000
    2.0000
    3.0000
    4.0000
    5.0000
newlongs =
    1.0000
    2.0000
    3.0000
    3.5000
    4.0000
    5.0000

```

In the original `lats`, there is a gap of 2 degrees between the 2 and the 4. A linearly interpolated point, (3,3.5) was therefore inserted in `newlats` and `newlongs`. Similarly, in the original `longs`, there is a gap of 2 degrees between the 1 and the 3. The point (1.5,2) was therefore interpolated and placed into `newlats` and `newlongs`. Now, no adjacent points in either `newlats` or `newlongs` is greater than `maxdiff` in separation.

The `interp` function returns the original data with new linearly interpolated points inserted. Sometimes, however, only the interpolated values are desired. The commands `intrplat` and `intrplon` provide a capability similar to MATLAB's `interp1` command, allowing for different methods of interpolation.

Use `intrplat` to interpolate a latitude for a given longitude. Given a monotonic set of longitudes and their matching latitude points, you can interpolate a new latitude for a given longitude in a linear, spline, cubic, rhumb line, or great circle sense.

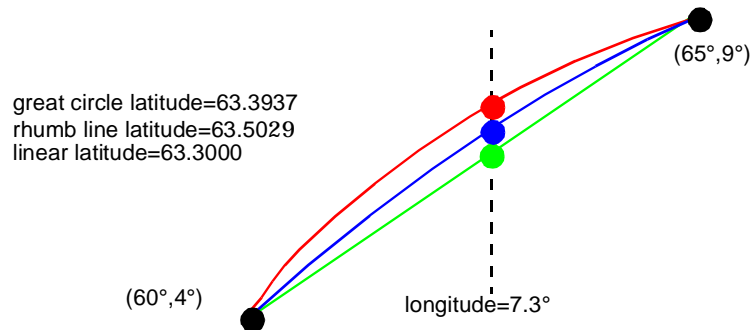
Here, find the latitude corresponding to a longitude of 7.3° in the following data in a linear, great circle, and rhumb line sense:

```
longs = [1 3 4 9 13]; lats = [57 68 60 65 56]; newlong = 7.3;

newlat = intrpl at(longs, lats, newlong, 'linear')
newlat =
    63.3000

newlat = intrpl at(longs, lats, newlong, 'gc')
newlat =
    63.5029

newlat = intrpl at(longs, lats, newlong, 'rh')
newlat =
    63.3937
```



The `intrpl on` function provides the same capability for interpolating new longitudes for given latitudes.

Polygon Area

For vector data in polygon format, geographic areas can be calculated using the command `areaint`. The function performs a numerical integration using Green's Theorem for the area on a surface enclosed by a polygon. Since this is a discrete integration on discrete data, the results are not exact. Nevertheless, the method provides the best means of calculating the area of arbitrarily shaped regions. Better measures result from better data.

The Mapping Toolbox function `areaint` (for area by integration), like the other area functions `areaquad` and `areamat`, returns areas as a fraction of the entire planet's surface, unless a radius is provided. Here we calculate the area of the continental United States using the `usl o` workspace. Three areas are returned, because the data contains three polygons: Long Island, Martha's Vineyard, and the rest of the continental U.S.:

```
load usl o
earthradius = almanac('earth', 'radius');
area = areaint(uslat, uslon, earthradius)
area =
    1.0e+06 *
        7.9256
        0.0035
        0.0004
```

Since the default Earth radius is in kilometers, the area is in square kilometers. From the same workspace, the areas of the Great Lakes can be calculated, this time in square miles:

```
earthradius = almanac('earth', 'radius', 'miles');
area = areaint(gtlakelat, gtlakel on, earthradius)
area =
    1.0e+04 *
        8.0124
        1.0382
        0.7635
```

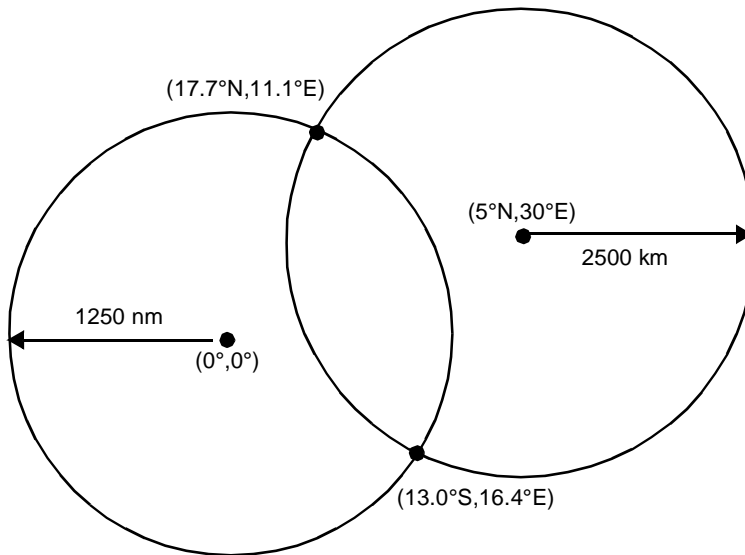
Again, three areas are returned, the largest for the polygon representing Superior, Michigan, and Huron together, the other two for Erie and Ontario.

Vector Calculations – Intersections

The Mapping Toolbox provides a set of functions to perform intersection calculations on vector data computed by the toolbox, which include great and small circles, as well as rhumb line tracks. The functions do not, however, determine intersections of arbitrary vector data.

Compute the intersection of a small circle centered at $(0^\circ, 0^\circ)$ with a radius of 1250 nautical miles and a small circle centered at $(5^\circ\text{N}, 30^\circ\text{E})$ with a radius of 2500 kilometers:

```
[lat, long] = scxsc(0, 0, nm2deg(1250), 5, 30, km2deg(2500))
lat =
    17.7487 -12.9839
long =
    11.0624 16.4170
```



Notice that, in general, small circles intersect twice or never. For the case of exact tangency, `scxsc` returns two identical intersection points. Other similar commands include `rhxrh` for intersecting rhumb lines, `gcxgc` for intersecting great circles, and `gcxsc` for intersecting a great circle with a small circle.

Imagine a ship setting sail from Norfolk, Virginia ($37^{\circ}\text{N}, 76^{\circ}\text{W}$), maintaining a steady due-east course (90°), and another ship setting sail from Dakar, Senegal ($15^{\circ}\text{N}, 17^{\circ}\text{W}$), with a steady northwest course (315°). Where would the tracks of the two vessels cross?

[lat, long] = rhxrh(37, -76, 90, 15, -17, 315)

lat =

37

long =

-41.7028

The intersection of the tracks is at ($37^{\circ}\text{N}, 41.7^{\circ}\text{W}$), which is roughly 600 nautical miles west of the Azores in the Atlantic Ocean.



Trimming Vector Data to a Defined Region

Sometimes vector data may extend beyond the geographic region of interest. For example, if you have world coastline data, but you require a map of Australia only, you may want to create new variables that contain only the data you need. You might also want to trim your data to save memory or to speed up calculations and display.

We can distinguish between trimming line data and patch data. Line data can be trimmed by simply removing points outside the region of interest. Patch data requires a more complicated method to ensure that the patch objects are correctly displayed. For the vector data, two functions are available to achieve this. If the variables are to be treated as line data, the `maptriml` command will return variables containing only those points in the defined region. If polygon format must be retained, the `maptrimp` command will ensure this, although compared to the line formatted data, the patch-trimmed data will be larger and take more time to compute.

It should be noted that for viewing purposes, trimming the data before display is not required. The vector data is automatically trimmed to the region specified by the frame limits (`FLatLimit` and `FLonLimit` map axes properties) for azimuthal projections and frame or map limits (`MapLatLimit` and `MapLonLimit` map axes properties) for non-azimuthal projections. The trimming is done internally in the display routine, so the original data remains intact. See “Map Axes” in Chapter 2 of this document, along with the reference pages for the trimming functions in the “Map Reference” section of the *Mapping Toolbox Reference Guide*, for further information on trimming geographic vector data.

Load the coast MAT-File, and trim the vector data to a region centered on Australia:

```
load coast
whos
```

Name	Size	Bytes	Class
lat	9589x1	76712	double array
long	9589x1	76712	double array

```
latlim = [-50 0]; longlim = [105 160];
[linelat, linelong] = maptriml(lat, long, latlim, longlim);
[polylat, polylong] = maptrimp(lat, long, latlim, longlim);
```

```
whos
```

Name	Size	Bytes	Class
lat	9589x1	76712	double array
latlim	1x2	16	double array
linelat	870x1	6960	double array
linelong	870x1	6960	double array
long	9589x1	76712	double array
longlim	1x2	16	double array
polylat	1020x1	8160	double array
polylong	1020x1	8160	double array



Simplifying Vector Data

Sometimes you may need to reduce the number of points in your vector data, while still maintaining an accurate representation of the geographical data. A quick and easy method would be to throw out every other element or set of elements in each vector. However, the result may be a very crude representation of the original data. Another option exists. The Mapping Toolbox provides a function implementing a powerful line simplification algorithm that selectively deletes points.

For example, reduce the number of points in the state of Massachusetts with the `reducem` function:

```
load usahi
whos
```

Name	Size	Bytes	Class
stateline	1x51	837656	struct array
statepatch	1x51	837758	struct array
statetext	1x51	51190	struct array

The `stateline` structure contains vector data for all 50 U.S. states and the District of Columbia. We are only interested in the state of Massachusetts, the 19th entry in the structure:

```
lat = stateline(19).lat;
long = stateline(19).long;
clear state*
```

Now simplify your vector data:

```
[newlat, newlong, cerr, tol] = reducem(lat, long);
whos
```

Name	Size	Bytes	Class
cerr	1x1	8	double array
lat	958x1	7664	double array
long	958x1	7664	double array
newlat	253x1	2024	double array
newlong	253x1	2024	double array
tol	1x1	8	double array

The Massachusetts vector data has been reduced to almost a quarter of its original size:

```
size(newlat)/size(lat)
ans =
    0.2641
```

Plots of the datasets, however, show two virtually identical maps:

original (958 points)



reduced (253 points)



Most of the reduction has taken place along the straight borders of western Massachusetts, which now consist of just a handful of points.

The function also determined an initial, default tolerance value `tol`, since one was not specified, as well as an error estimate of polygon arc lengths `cerr` between the two sets of data. The tolerance value, which is defined as the maximum allowable deviation from a straight line approximation, is 0.006 degrees (in the same angular units as the vector data), and the calculated error in arc length is 3.3%:

```
tol
tol =
    0.0060

cerr
cerr =
    0.0331
```

The original vector data can be reduced even further by increasing the tolerance level. If `tol` is increased to 0.02 degrees, the reduction is now almost tenfold with an error of about 7.7%:

```
[newlat, newlong, cerr] = reducec(lat, long, 0.02);
whos
```

Name	Size	Bytes	Class
cerr	1x1	8	double array
lat	958x1	7664	double array
long	958x1	7664	double array
newlat	109x1	872	double array
newlong	109x1	872	double array
tol	1x1	8	double array

```
size(newlat)/size(lat)
ans =
    0.1138

cerr
cerr =
    0.0768
```


Here is the displayed map:

reduced (109 points)



There is now a noticeable degradation of the line vector map along the eastern coast of the state. Nonetheless, it still retains much detail, especially considering the enormous reduction of data.

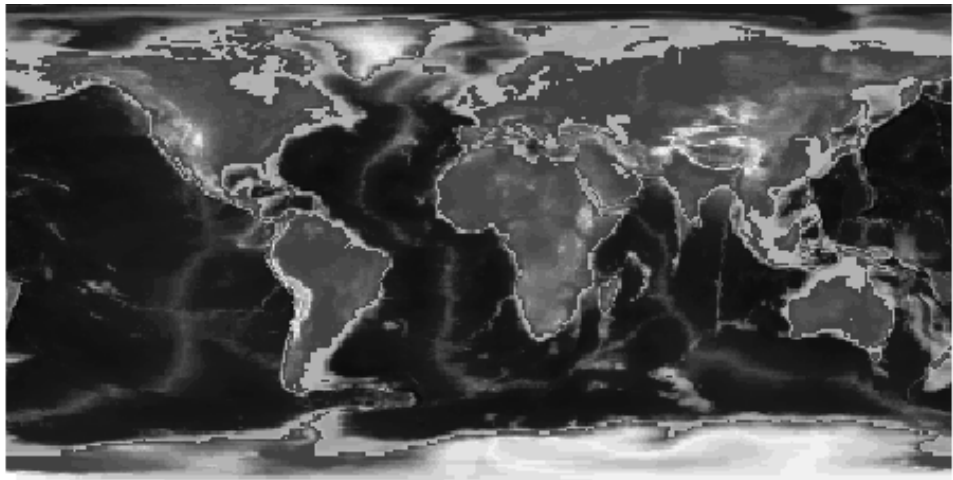
NOTE: Reduced geographic data may not always be appropriate for display. If all intermediate points in a dataset are reduced, then lines appearing straight in one projection will be incorrectly displayed as straight lines in others.

Working with Matrix Maps

One of the most powerful features of the Mapping Toolbox is its capacity for manipulating matrix maps. Two types of matrix maps are defined by the Mapping Toolbox: regular and general. Regular matrix maps are rectangular in appearance and are a subset of general matrix maps, which can be of practically any shape or orientation.

Regular Matrix Maps

A *regular matrix map* is one in which columns of data run south to north, rows of data run west to east, and each matrix element represents the same angular step in each direction, for all rows and columns. For example, if each row represents one degree of latitude, and each column one degree of longitude, and the matrix is aligned north and south, then it is a regular matrix map. The topo map provided with MATLAB is such a regular matrix map:



The Map Legend

Many regular matrix maps do not cover the entire planet. It is therefore necessary to identify the scale and placement of the map with a special data structure, the *map legend*. The map legend is a three-element vector that identifies the size of each matrix entry and the coordinates on the globe of the northwestern corner of the map. The three entries are given in terms of angular units of degrees only. The specific format of the structure is as follows:

$$\text{map legend} = [\text{cells/angle unit} \quad \text{north-latitude} \quad \text{west-longitude}]$$

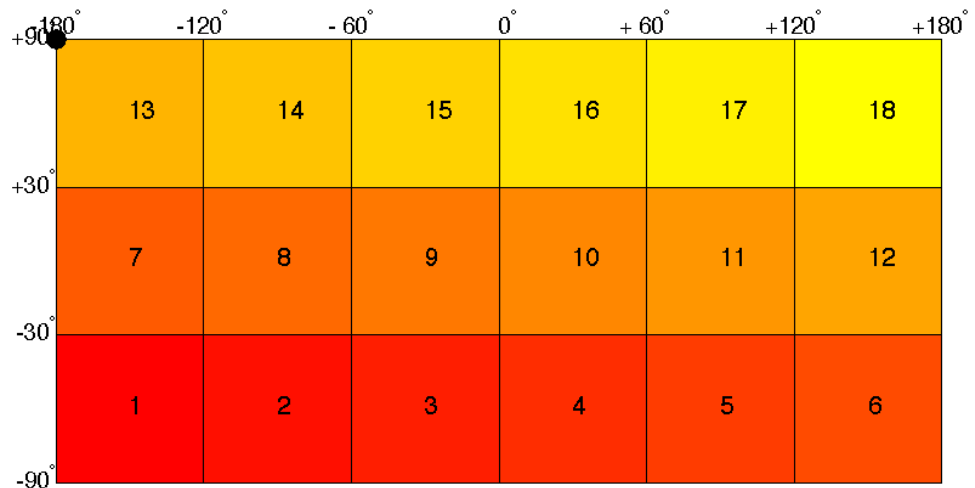
For the topo matrix map, each matrix cell represents one degree, the northern edge is the North Pole, and the western edge is the Prime Meridian. Therefore, the map legend for topo is:

$$\text{topo legend} = [1 \quad 90 \quad 0]$$

This map legend structure is stored in the topo workspace. Imagine an extremely coarse map of the world in which each cell represents 60°. Such a map matrix would be 3-by-6, and its map legend would be:

$$\text{map legend} = [1/60 \quad 90 \quad -180] = [0.0167 \quad 90 \quad -180]$$

In this case, the map's western edge is the International Date Line, at 180°W:



Note that the first row of the matrix is displayed as the bottom of the map, while the last row is displayed as the top. All regular matrix maps in the Mapping Toolbox, as well as regular surfaces in MATLAB, are displayed in this fashion.

As an example of a matrix map that does not encompass the entire world, load the Cape Cod image using the `loadcape` script.

```
loadcape
maplegend
maplegend =
    120    44   -72
```

The `maplegend` indicates that there are 120 cells per angular degree. This has a horizontal resolution 120 times finer than that of the `topo` matrix map, which was one cell per degree. The cape region covered here extends from 41°N to 44°N, and from 72°W to 69°W.

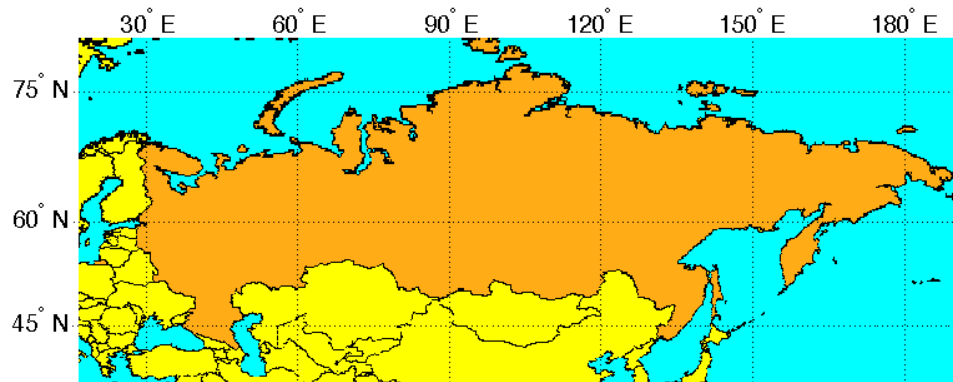
```
[latlimits, longlimits] = limitm(map, maplegend)
latlimits =
    41    44
longlimits =
   -72   -69
```

The Geographic Meaning

A matrix map and its associated map legend can provide a host of geographic information regarding the map and its entries. First, as was just demonstrated, the north, south, east, and west limits of the mapped area can be determined as follows:

```
load russi a
[latlim, longlim] = limitm(map, maplegend)
latlim =
    35    80
longlim =
    15   190
```

The map in the `ruusia` workspace extends over the International Date Line (180° longitude). You could use the previously described command `np12pi` to rename the eastern limit to be -170° , or 170° W.



The command `setlatlon` allows you to determine the geographic coordinates of a particular matrix element. The returned coordinates actually show the center of the geographic area represented by the matrix entry:

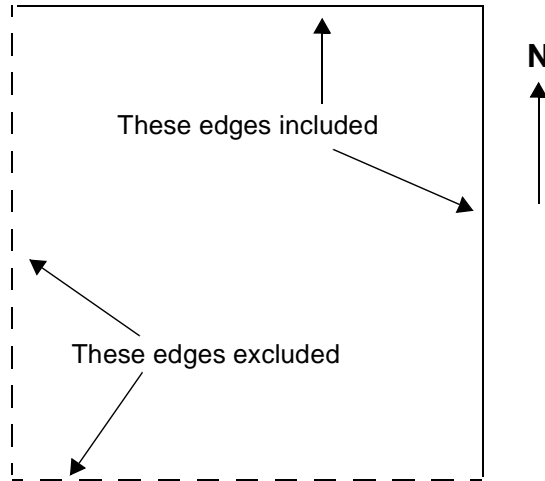
```
row = 23; col = 79;
[lat, long] = setlatlon(map, maplegend, row, col)
lat =
    39.5000
long =
    30.7000
```

You can also determine the row and column of the matrix map entry containing a given point:

```
[r, c] = setpostn(map, maplegend, lat, long)
r =
    23
c =
    79
```

Each matrix entry can be thought of as an angular “square” which includes its northern and eastern edges, but not its western and southern edges:

A Matrix Map Entry



The exceptions to this are that the southernmost row (row 1) also contains its southern edge, and the westernmost column (column 1) contains its western edge, except when the map encompasses the entire 360° of longitude. In that case, the westernmost edge of the first column is not included, because it is identical to the easternmost edge of the last column. These exceptions ensure that all points on the globe can be represented exactly once in a regular matrix map.

Although each map matrix entry represents an area, not a point, it is often convenient to assign singular coordinates for reference. An entry is defined by the point in the center of the area represented by the entry. For example, reference the center cell coordinate for the 3rd row, 17th column, of the Russia map:

```
row = 3; col = 17;
[lat, long] = setlatln(map, maplegend, row, col)
lat =
    35.5000
long =
    18.3000
```

Since the cells in the Russia matrix map represent 0.2° squares (5 cells per degree), the cell in question extends from 35.6°S to 35.4°S and from 18.2°E to 18.4°E.

Accessing Matrix Map Elements

The actual values contained within the map matrix entries are important as well. The Mapping Toolbox provides several functions for accessing and altering the values of matrix map entries.

If the actual row and column of a desired entry are known, then a simple matrix index can return the appropriate value. Use the row and column from the last example (3rd row, 17th column) to determine the value of that cell:

```
value = map(row, col)
value =
    2
```

More often, the geographic coordinates are known:

```
value = latln2val(map, maplegend, lat, long)
value =
    2
```

The latitude-longitude coordinates associated with particular values in a matrix map can be found with `findm`, which is similar to the standard MATLAB command `find`. Here we find the coordinates in the `topo` map with values greater than 5500 meters:

```
load topo
[lat, long] = findm(topo>5500, topolegend);
[lat long]
ans =
    34.5000    79.5000
    34.5000    80.5000
    30.5000    84.5000
    28.5000    86.5000
```

You can change all instances of a given value in a matrix map to a new value in one step. Here is an example using the map of Russia:

```
oldcode = ltl2val(map, maplegend, 37, 79)
oldcode =
    4
newmap = changem(map, 5, oldcode);
newcode = ltl2val(newmap, maplegend, 37, 79)
newcode =
    5
```

All entries in `newmap` corresponding to 4s in `map` now have the value 5. You can also define a mask to determine map entries to change. A mask is a matrix the same size as the map matrix, with 1s everywhere the entry is to take on the new value. A mask is often generated by a logical operation on a map variable, a topic which is described in greater detail below. The map in the `ruusia` workspace contains 3s for Russia. If you want to set every non-Russia matrix entry to zero, you can use `maskm`:

```
nonruusia = (map~=3);
newmap = maskm(map, nonruusia, 0);
newcode = ltl2val(newmap, maplegend, 37, 79)
newcode =
    0
```


Finally, if you know the latitude and longitude limits of a region, you can calculate the required matrix size and an appropriate map legend for a desired map scale. Before making a large, memory-taxing matrix map, you should first determine its size. For a map of the continental United States at a scale of 10 cells per degree, do the following:

```
scale = 10;
[r, c, maplegend] = size([25 55], [-130 -60], scale)
r =
    300
c =
    700
maplegend =
    10    55   -130
```

This matrix map would be 300-by-700. What if the scale were reduced to 5 rows/columns per degree?

```
scale2 = 5;
[r, c, maplegend] = size([25 55], [-130 -60], scale2)
r =
    150
c =
    350
maplegend =
     5    55   -130
```

A 150-by-300 matrix might be more manageable.

Valued Maps and Indexed Maps

A *valued map* is a matrix map in which the entries represent some value or measurement. A simple example of a valued map is the `topo` matrix in the `topo` workspace. Each entry of `topo` is an average elevation in meters for that piece of the Earth. The entries of a valued map answer the question “How much is here?”

An *indexed map* is a matrix map in which the entries are an index value indicating where more information might be found. A simple example of an indexed map is the `map` variable in the `worldmtx` workspace. Each entry of `map` is an index to the `nations` structure indicating which nation contains that piece of the Earth. The entries of an indexed map answer the question “What is this?”

To illustrate the difference between valued maps and indexed maps, consider two geographic points, (45°N,105°E) and (47°N,108°E). The valued map `topo` contains a different kind of information for these points from what is found in the indexed world map:

```
load topo
ltn2val(topo, topl legend, 45, 105)
    1270
ltn2val(topo, topl legend, 47, 108)
ans =
    1513
```

As might be expected, the two regions represented by the appropriate entries of `topo` have different average elevation values. However, they have the same index value in `map`, because they are in the same country:

```
load worldmtx
ltn2val(map, map legend, 45, 105)
ans =
    119
ltn2val(map, map legend, 47, 108)
ans =
    119

nations(119).name
ans =
Mongolia
```

Matrix Maps as Logical Variables

A powerful use of matrix maps is the application of logical conditions to create *logical maps*. Logical maps are matrix maps consisting entirely of 1s and 0s. They can be created by performing logical tests on matrix map variables. The resulting binary matrix is the same size as the original map(s) and can use the same map legend, as the following illustrates:

```
l o g i c a l m a p = ( r e a l m a p > 0 )
```

Multiple conditions can operate on the same map:

```
l o g i c a l m a p = ( r e a l m a p > - 1 0 0 ) & ( r e a l m a p < 1 0 0 )
```

Or, if several maps are the same size and share the same map legend, a logical map can be created by testing conditions on several “layers” of maps:

```
l o g i c a l m a p = ( p o p u l a t i o n > 1 0 0 0 0 ) & ( e l e v a t i o n < 4 0 0 ) & . . .  
                  ( c o u n t r y == n i g e r i a )
```

The Mapping Toolbox provides functions that enable the creation of logical maps. For example, the 5-cell-per-degree maps covering the continental United States of all 1s and all 0s can be easily constructed:

```
l a t l i m s = [ 2 5 5 5 ]; l o n g l i m s = [ - 1 3 0 - 6 0 ]; s c a l e = 5;  
o n e s m a p = o n e m ( l a t l i m s , l o n g l i m s , s c a l e );  
z e r o e s m a p = z e r o m ( l a t l i m s , l o n g l i m s , s c a l e );
```

Maps of all NaNs and sparse maps of all 0s can be made in a similar fashion with the commands `nanm` and `spzerom`.

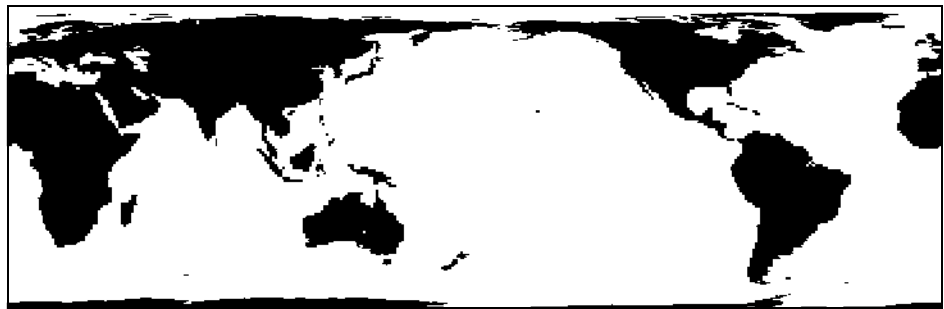
A useful way of analyzing the results of logical map manipulations is to determine the area satisfying the conditions (i.e., with a logical value of 1). The command `areamat` can provide the fractional surface area on the globe associated with 1s in a logical map. Each matrix element is a quadrangle, and the sum of the areas meeting the logical condition provides the total area. For example, according to the `topo` map, what fraction of the Earth lies above sea level?

```
l o a d t o p o  
a = a r e a m a t ( ( t o p o > 0 ) , t o p o l e g e n d )  
a =  
    0. 2890
```

The answer is about 30%. If a planetary radius is provided in, for example, kilometers, the total physical area above 0 elevation can be returned in square kilometers:

```
a = areamat((topo>0), toplegend, almanac('earth', 'radius'))  
a =  
    1.4739e+08
```

To interpret this graphically, remember that the logical matrix map (`topo>0`) is binary. The following is a display of the map with black where the condition is true (i.e., the logical map entry is “1”) and white where it is false (“0”). It is displayed in an Equal-Area Cylindrical projection, which vertically shrinks rows near the poles to show their true relative area. Similarly, the `areamat` command “shrinks” the contribution of these rows to the total area.



General Matrix Maps

In addition to regular matrix maps, the Mapping Toolbox provides another matrix map format: *general matrix maps*. These maps can be displayed, and their values and coordinates can be queried, but much of the functionality available for regular matrix maps cannot be exploited for general matrix maps.

The examples thus far have shown maps that covered simple, regular quadrangles, that is, geographically rectangular. General matrix maps, in addition to these rectangular orientations, can have other shapes as well.

The Map Format

To define a general matrix map, you need three variables: the matrix of indices or values associated with the mapped region, a matrix giving cell-by-cell latitude coordinates, and a matrix giving cell-by-cell longitude coordinates.

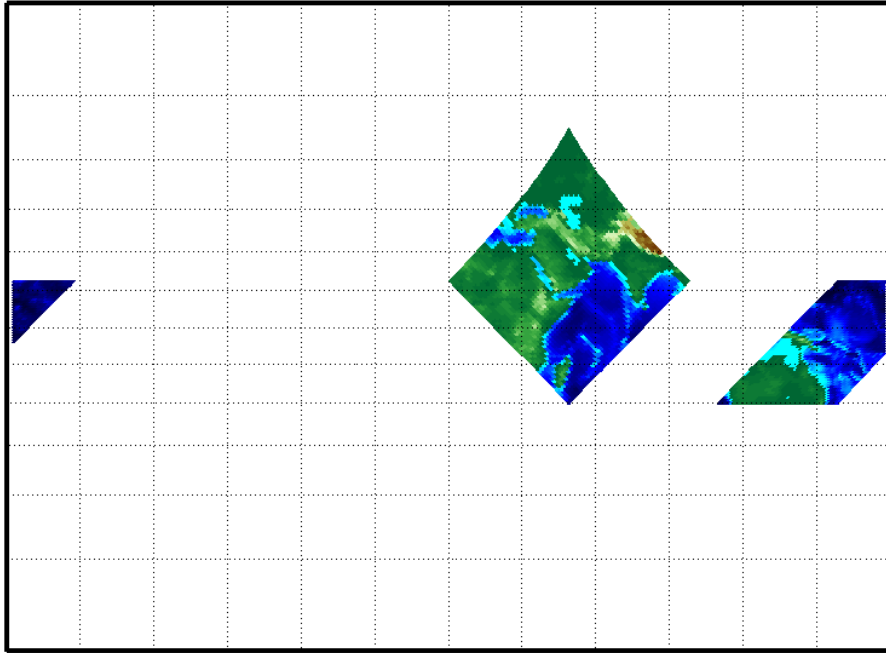
An example of an irregularly shaped general matrix map is available in the Mapping Toolbox for examination:

```
load mapmtx
whos
```

Name	Size	Bytes	Class
lg1	50x50	20000	double array
lg2	50x50	20000	double array
lt1	50x50	20000	double array
lt2	50x50	20000	double array
map1	50x50	20000	double array
map2	50x50	20000	double array

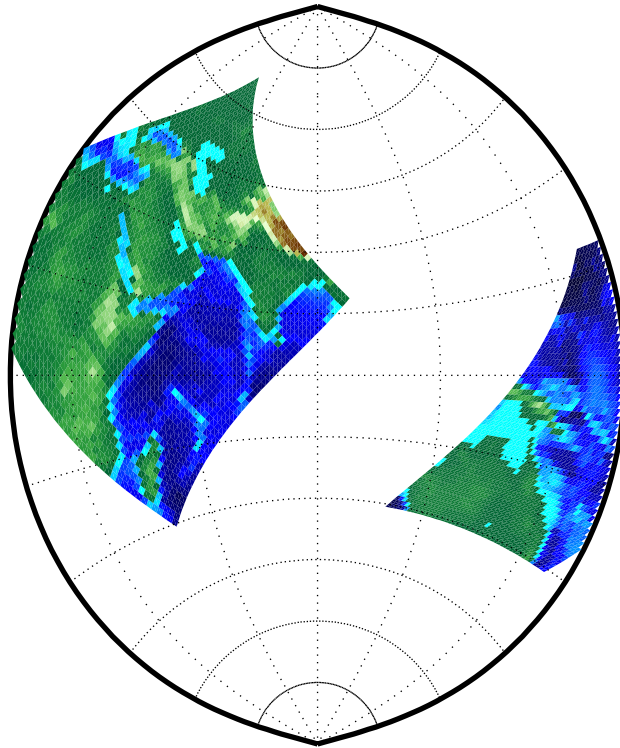
Two general matrix maps are in this workspace, each requiring three variables. The values contained in map1 correspond to the latitude and longitude coordinates, respectively, in lt1 and lg1. Notice that all three matrices are the same size. Similarly, map2, lt2, and lg2 together form a second general matrix map. These datasets were extracted from the topo matrix map shown in previous examples. Neither of these maps is regular in that their columns do not run north-and-south.

To get an idea of their geography, display them together:



Notice that neither map is a regular rectangle. One looks like a diamond geographically, the other like a trapezoid. The trapezoid is displayed in two pieces because it crosses the edge of the map. These shapes can be thought of as the geographic organization of the data, just as rectangles are for regular matrix maps. But, just as for regular matrix maps, this organizational logic does not mean that displays of these maps are necessarily a specific shape.

Here you can view these maps together in a Polyconic projection:



Since the Polyconic is limited to a 150° range in longitude, those portions of the maps outside this region are trimmed automatically.

Geographic Interpretations

The Mapping Toolbox supports several different interpretations of general matrix maps. First, a map matrix having the same size as the latitude and longitude coordinate matrices represents the values of the map data at the corresponding geographic points. Next, a map matrix having one fewer row and column than the geographic coordinate matrices represents the values of the map data within the area formed by the four adjacent latitudes and longitudes. Finally, if the latitude and longitude matrices are still smaller than the map matrix, you can interpret them as describing a coarse *graticule*, or mesh of latitude and longitude cells into which the blocks of map data are warped.

This section discusses the first two interpretations of general matrix maps. Chapter 2, “Displaying Maps,” has more information on the use of graticules.

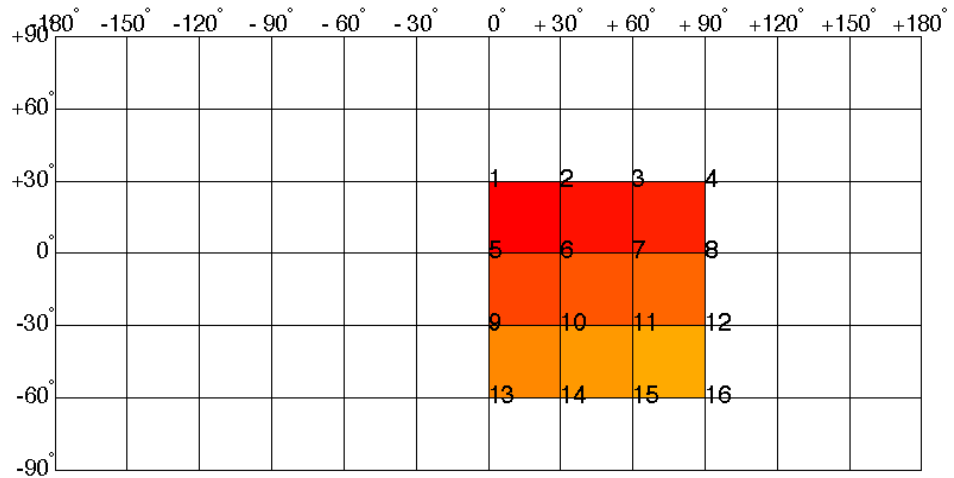
As an example of the first interpretation, consider a 4-by-4 map matrix whose cell size is 30-by-30 degrees, along with its corresponding 4-by-4 latitude and longitude matrices:

```
map = [ 1  2  3  4; ...
        5  6  7  8; ...
        9 10 11 12; ...
       13 14 15 16];

lat = [ 30  30  30  30; ...
        0   0   0   0; ...
       -30 -30 -30 -30; ...
       -60 -60 -60 -60];

long = [ 0 30 60 90; ...
         0 30 60 90; ...
         0 30 60 90; ...
         0 30 60 90];
```


This general matrix map is displayed with the values of map shown at the associated latitudes and longitudes:

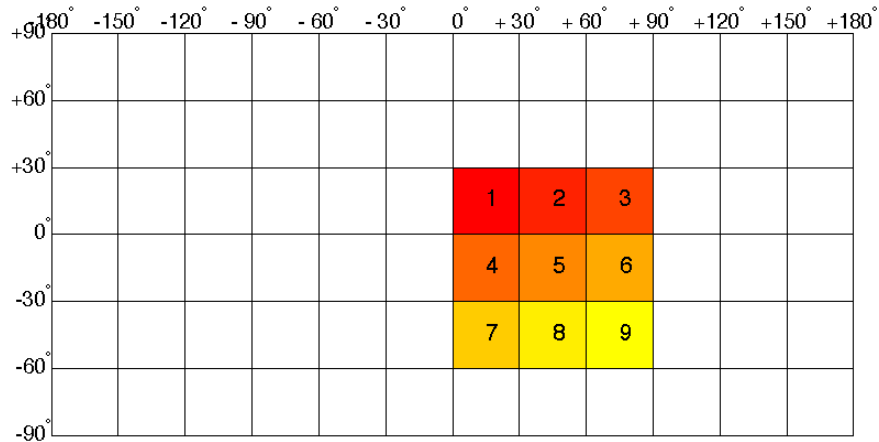


Notice that only 9 of the 16 total cells are displayed. The value displayed for each cell is the value at the upper left corner of that cell, whose coordinates are given by the corresponding `lat` and `long` elements. By MATLAB convention, the last row and column of the map matrix is not displayed, although they exist in the `CData` property of the surface object.

For the second interpretation, consider a 3-by-3 map matrix with the same `lat` and `long` variables:

```
map = [ 1  2  3; ...
       4  5  6; ...
       7  8  9];
```

Here is a surface plot of the map matrix, with the values of map shown at the center of the associated cells:



All of the map data is displayed for this general matrix map. The value of each cell is the value at the center of the cell, and the latitudes and longitudes in the coordinate matrices are the boundaries for the cells.

You may have noticed that the first row of the matrix is displayed as the top of the map, whereas for a regular matrix map, the opposite was true; the first row corresponded to the bottom of the map. This difference is due to the lat and long matrices and how they are ordered. The lat and long coordinate matrices determine the arrangement of the general matrix map.

Of course, a regular matrix map can also be considered a general matrix map. All that are required are the graticule or coordinate matrices, which can be produced from the regular matrix map itself using the `meshgrat` function:

```
load topo
[lat, lon] = meshgrat(topo, topol legend);
whos
```

Name	Size	Bytes	Class
lat	180x360	518400	double array
lon	180x360	518400	double array
topo	180x360	518400	double array
topol legend	1x3	24	double array
topomap1	64x3	1536	double array
topomap2	128x3	3072	double array

Displaying Maps

Introduction to Mapping Graphics	2-2
Map Axes	2-3
Accessing and Manipulating Map Axes Properties	2-6
The Map Frame	2-11
The Map Grid	2-15
Map and Frame Limits	2-18
Switching Between Projections	2-19
Projected and Unprojected Objects	2-22
Displaying Vector Maps as Lines	2-24
Displaying Vector Maps as Patches	2-27
Displaying Matrix Map	2-30
The Graticule	2-31
Coloring Matrix Maps	2-34
Data Representation	2-36
Mapped Light Objects	2-40
The Geographic Data Structure	2-43
Interacting with Displayed Maps	2-47
Specialized Map Displays	2-50
Thematic Map Functions	2-50
Choropleth Maps	2-52

Introduction to Mapping Graphics

One of the fundamental concepts underlying the Mapping Toolbox is the idea that maps are variables that exist independently of their display. However, for visual geographic data analysis, application development, and presentation of results, quality display capabilities are a must.

With the Mapping Toolbox, you can display geographic information almost as easily as you can plot regular data in MATLAB. Most mapping commands are similar to MATLAB commands, except they deal with latitude and longitude spherical coordinates instead of x and y Cartesian coordinates. Mapping commands typically have the same names as their MATLAB counterparts, with the addition of an 'm,' for maps, at the end. For example, the Mapping Toolbox analog to MATLAB's `plot` command is `plotm`.

The Mapping Toolbox manages most of the steps in displaying a map. It will project your data, cut and trim it to desired limits, and display the resulting map for you. The toolbox also provides the means to add other elements to your displayed map, such as a frame, gridlines, coordinate labels, and text labels. If you change your projection properties, or even the projection itself, the Mapping Toolbox will redraw the map with the new settings, undoing any cuts or trims if necessary. See Chapter 4, "Map Projections" for information on how to project data without displaying it.

The toolbox also makes it easy to modify and manipulate maps. The map display and mapped objects can be modified either from the command line or through graphical user interfaces and property editing tools that can be invoked by clicking on the display. Most mapping display commands have graphical interfaces. See Chapter 6, "GUI Tools" for more on these capabilities.

NOTE: An important restriction accompanies this ease of use. The Mapping Toolbox manages the map display with the `UserData` property field in the Axes control. The `UserData` property of mapped objects is also used by the toolbox. This may cause conflicts with other functions that use the `UserData` property field.

Map Axes

To display maps of any kind in the Mapping Toolbox, the user must first define a map axes. A *map axes* is a normal axes that has been prepared for mapping using the `axesm` command that defines the map projection properties. Before you execute any mapping command, you need to define a map axes in the current MATLAB axes and choose a map projection using `axesm`.

NOTE: To successfully employ any map display function, such as `plotm`, `fillm`, or `surfacem`, the current axes must be a map axes as created by `axesm`. An axes is considered a valid map axes if its `UserData` property contains a structure defining a map projection.

A *projection* is the systematic means of representing three-dimensional geographic data in two dimensions, a topic that is covered more extensively in Chapter 4, “Map Projections.” Here, it is sufficient to understand that many different projections are available, each with different features. Some projections maintain correct relative area between objects, for example, or correct angular relationships. Some are well-suited to displaying features at certain latitudes or with certain orientations. The command `maps` displays a list of available projections.

When creating a map axes object, you must define its projection by using an appropriate ID string. All other properties can be set by default. For example, the following are all valid commands for creating a Miller projection map axes:

```
axesm miller
axesm('miller')
axesm('MapProjection','miller')
```

Other properties can be set for the map axes during creation as well:

```
axesm('MapProjection','mercator',...
      'Frame','on',...
      'AngleUnits','degrees',...
      'FontName','courier',...
      'parallel label','on',...
      'Origin',[0 180 0])
```

If you specify some map axes parameters with `axesm`, the Mapping Toolbox determines appropriate values for other related map axes properties. As an example, create a map axes with an Equal-Area Conic projection for part of the world:

```
axesm('MapProjection', 'eqaconic', ...  
      'MapLatLimit', [30 60], ...  
      'MapLonLimit', [-15 45], ...  
      'MapParallels', [])
```

The `axesm` function will automatically set the origin and frame limits for a map centered on the limited geographic region, with the results of subsequent mapping display commands shown only within the specified limits. Empty matrices are used to force recalculation of a property or a return to the default value. For example, setting the map parallels to an empty matrix will automatically determine map parallels appropriate for the current map limits, minimizing distortion in the projection.

Since the `MapProjection` property is the only property that must be explicitly set, it can be specified without the property name, as long as the projection name is the first argument in `axesm`. For instance:

```
axesm('miller', 'PropertyName1', PropertyValue1, ...  
      'PropertyName2', PropertyValue2, ...)
```

The `axesm` command without any arguments activates a GUI that allows for easy manipulation of the map axes properties.

For proper map appearance and ease of use, the Mapping Toolbox changes some of the default MATLAB axes properties. The altered properties are:

- The `NextPlot` property is set to 'add'.
- The `UserData` property contains the map projection structure that defines the specific map axes properties.
- The `Box` property is set to 'on'.
- The `ButtonDownFcn` property is set to the function `ui_map_tbx`, which processes callbacks for Mapping Toolbox objects.
- The command `showaxes off` is executed to remove axes ticks from the MATLAB Cartesian axes (i.e., the `XTick`, `YTick`, and `ZTick` properties are set to empty matrices).

More information on MATLAB axes properties can be found in the *Using MATLAB Graphics* manual under “Axes.” The map axes properties are defined under the `axesm` heading in the “Mapping Reference” chapter in the *Mapping Toolbox Reference Guide*, while information on the use of the `axesm` GUI can be found in the “GUI Reference” chapter in the *Mapping Toolbox Reference Guide*.

Accessing and Manipulating Map Axes Properties

Just as the properties of the underlying standard axes can be accessed and manipulated using the MATLAB commands `set` and `get`, map axes properties can also be accessed and manipulated using the commands `setm` and `getm`.

NOTE: Use the `axesm` command to *create* a map axes. Use the `setm` command to *modify* the map axes.

As an example, create a map axes:

```
axesm('MapProjection','miller','Frame','on')
```

Suppose you would like to make the frame lines bordering the map thicker. First you need to determine the current line width of the frame. Access the property value by typing:

```
getm(gca,'FLineWidth')
ans =
     2
```

Try resetting the line width to four points:

```
setm(gca,'FLineWidth',4)
```

You can set any number of properties simultaneously with `setm`. Reduce the line width, change the projection to an Equidistant Cylindrical, and verify the changes:

```
setm(gca,'FLineWidth',3,'MapProjection','eqdcyl in')
```

```
getm(gca,'FLineWidth')
ans =
     3
```

```
getm(gca,'MapProjection')
ans =
eqdcyl in
```

To inspect the entire set of map axes properties at their current settings, use the following command:

```
getm(gca)
ans =
    mapprojection: 'eqdcyl in'
      angleunits: 'degrees'
        aspect: 'normal'
    fixedorient: []
        geoid: [1 0]
    maplatlimit: [-90 90]
    maplonlimit: [-180 180]
    mapparallels: 30
      nparallels: 1
        origin: [0 0 0]
      trimlat: [-90 90]
      trimlon: [-180 180]
        frame: 'on'
        ffill: 100
    fedgecolor: [0 0 0]
    ffacecolor: 'none'
      flatlimit: [-90 90]
    flinewidth: 3
      flonlimit: [-180 180]
        grid: 'off'
      galtitude: Inf
        gcolor: [0 0 0]
      glinestyle: ':'
      glinewidth: 0.5000
    mlineexception: []
      mlinefill: 100
      mlinelimit: []
    mlineolocation: 30
      mlinevisible: 'on'
    plineexception: []
      plinefill: 100
      plinelimit: []
    plineolocation: 15
      plinevisible: 'on'
      fontangle: 'normal'
```

```
fontcolor: [0 0 0]
fontname: 'helvetica'
fontsize: 9
fontunits: 'points'
fontweight: 'normal'
labelformat: 'compass'
labelunits: 'degrees'
meridianlabel: 'off'
mlabellocation: 30
mlabelparallel: 90
mlabelround: 0
parallellabel: 'off'
plabellocation: 15
plabelmeridian: -180
plabelround: 0
```

Similarly, the possible property values and their defaults can be displayed with the `setm` command alone:

```
setm(gca)

AngleUnits          [ {degrees} | radians | dms | dm ]
Aspect              [ {normal} | transverse ]
FixedOrient         FixedOrient is a read-only property
Geoid
MapLatLimit
MapLonLimit
MapParallels
MapProjection
NParallels          NParallels is a read-only property
Origin
TrimLat             TrimLat is a read-only property
TrimLon             TrimLon is a read-only property
Frame               [ on | {off} ]
FEdgeColor
FFaceColor
FFill
FlatLimit
FLineWidth
FLonLimit
Grid                [ on | {off} ]
GAltitude
GColor
GLineStyle          [ - | -- | -. | {:} ]
GLineWidth
MLineException
MLineFill
MLineLimit
MLineLocation
MLineVisible       [ {on} | off ]
PLineException
PLineFill
PLineLimit
PLineLocation
PLineVisible       [ {on} | off ]
FontAngle           [ {normal} | italic | oblique ]
```

FontCol or	
FontName	
FontSi ze	
FontUni ts	[inches centimeters normalized {points} pixels]
FontWei ght	[{normal} bold]
Label Format	[{compass} signed none]
Label Uni ts	[{degrees} radians dms dm]
Meri di anLabel	[on {off}]
MLabel Locati on	
MLabel Paral lel	
MLabel Round	
Paral lel Label	[on {off}]
PLabel Locati on	
PLabel Meri di an	
PLabel Round	

Properties can also be displayed individually:

```
setm(gca, 'AngleUnits')
AngleUnits      [ {degrees} | radians | dms | dm ]
```

```
setm(gca, 'MapProjection')
```

An axes's "MapProjection" property does not have a fixed set of property values.

```
setm(gca, 'Frame')
Frame           [ on | {off} ]
```

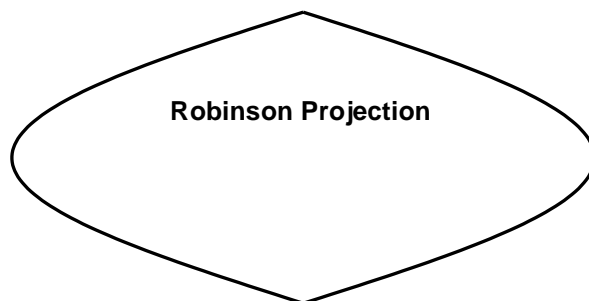
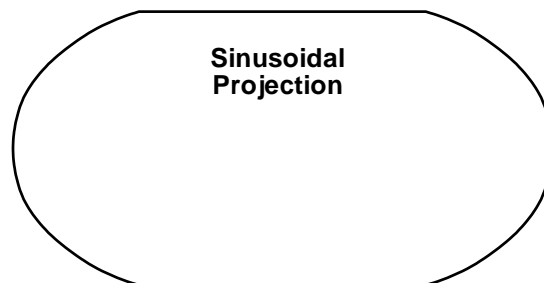
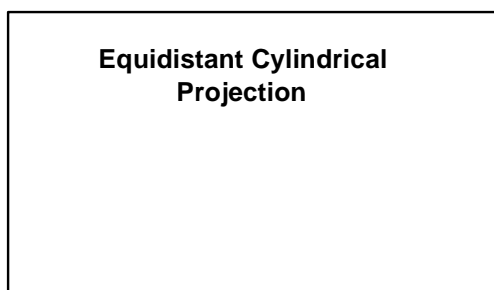
```
setm(gca, 'FixedOrient')
```

FixedOrient is a read-only property

The Map Frame

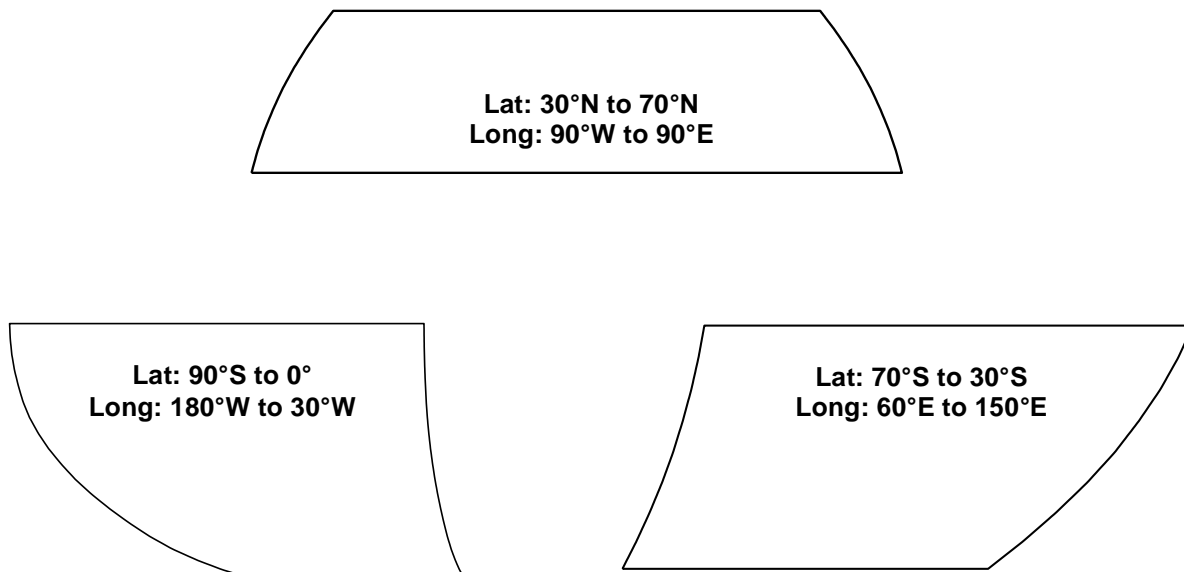
In the Mapping Toolbox, the *map frame* is a “box” enclosing the geographic display. The frame is displayed if the map axes property `Frame` is set to 'on'. This can be accomplished upon map axes creation with `axesm`, with `setm`, or with the direct command `framem on`. The frame is geographically defined as a latitude-longitude quadrangle that is projected appropriately. For example, on a map of the world, the frame might extend from pole to pole and a full 360° in longitude. In appearance, the frame would take on the characteristic shape of the projection. The examples below are full-world frames shown in three very different projections:

Full-World Map Frames



As a map object, each of the previously displayed frames is identical; however, the selection of a display projection has varied their appearance. Since each of the examples shows the entire world, `FLatLimit` is `[-90 90]`, and `FLonLimit` is `[-180 180]` for each case. The frame quadrangle can encompass smaller regions, as well:

Frame Quadrangles Shown in the Robinson Projection (Symmetric about Prime Meridian)



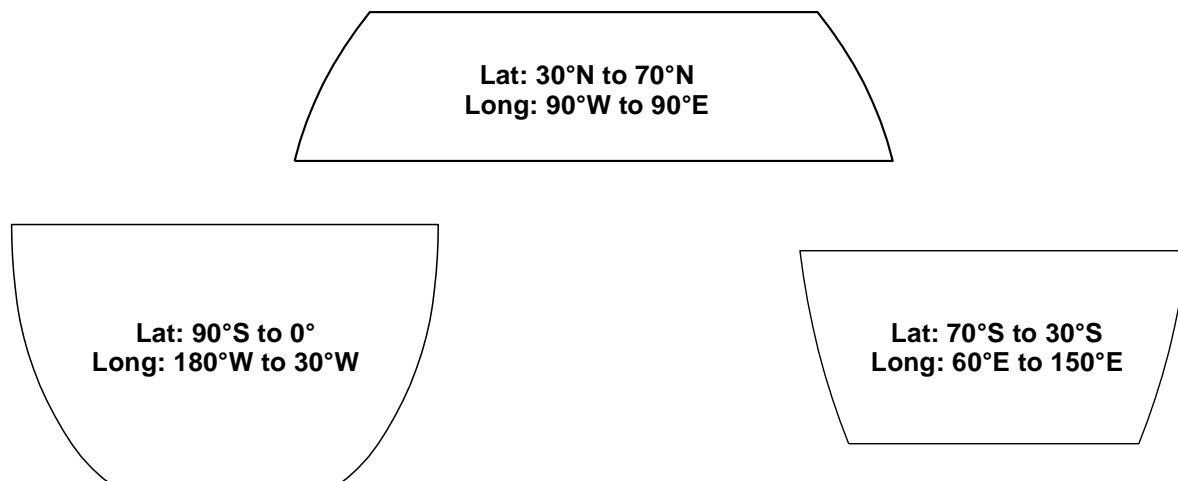
For the frames shown above, the projection is centered on the Prime Meridian, or 0 longitude. Such a frame would be the result of creating a map axes with the defaults for the Robinson projection and then resetting the frame limits to cover just part of the world.

For example, to view the asymmetric frame in the lower right of the previous figure, type:

```
axesm robinson
setm(gca, 'FLatLimit', [-70 -30], ...
      'FLonLimit', [60 150], ...
      'Frame', 'on')
```


When you want your frame to be symmetric about the region of interest, have `axesm` determine the proper settings for you. If you specify the map limits without specifying the map origin and frame limits, `axesm` will automatically set the appropriate values for a proper symmetric frame.

Frame Quadrangles Shown in the Robinson Projection (Symmetric about Map Limits)



For example, to view the symmetric frame in the lower right of the previous figure, set the map limits with `axesm`:

```
axesm('MapProjection', 'robinson', ...
      'MapLatLimit', [-70 -30], ...
      'MapLonLimit', [60 150], ...
      'Frame', 'on')
```

In addition to the latitude and longitude limits of the frame, other properties can be manipulated. The frame is actually a patch with a default face color set to 'none' and a default edge color of black. These map axes properties can be altered by manipulating the `FFaceColor` and `FEdgeColor` properties. For example, the command:

```
setm(gca, 'FFaceColor', 'cyan')
```

can make the background region of your display resemble water. Since the frame patch is always the lowest layer of a map display, other patches, perhaps representing land, will appear above the “water.” If an object is subsequently plotted “below” the frame patch, the frame altitude can be recalculated to lie below this object with the command `framem reset`. The frame is replaced and not reprojected.

The line width of the edge, which is 2 points by default, can be set using the `FLineWidth` property.

The primary advantage of displaying the map frame is that it can provide positional context for other displayed map objects. For example, when vector data of the coasts is displayed, the frame provides the “edge” of the world.

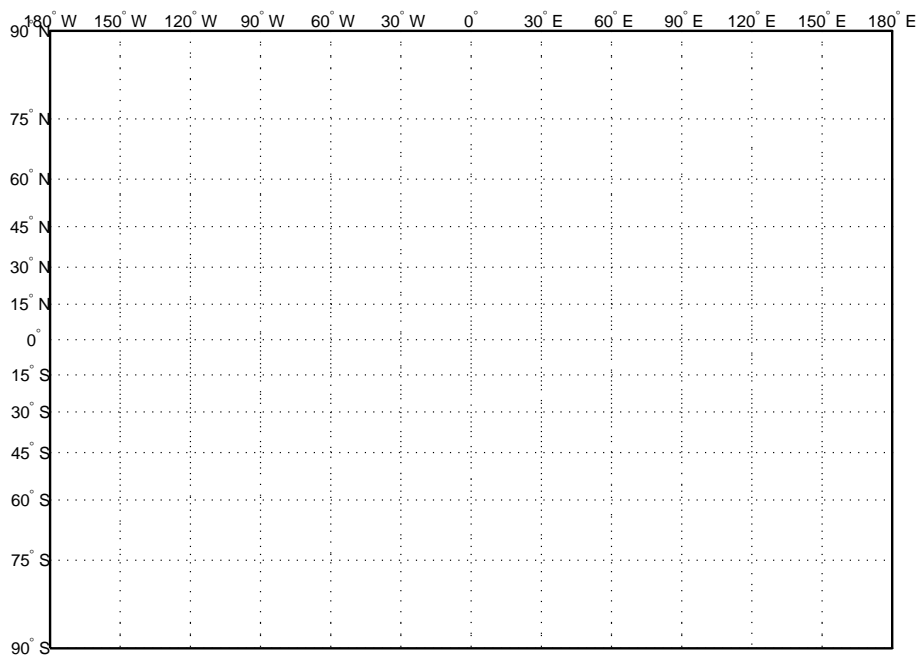
In addition to establishing frame properties upon map axes creation, or through the `setm` command, you can use the `framem` command. The command `framem` alone is a toggle for the `Frame` property, which controls the visibility of the frame. You can also call `framem` with property names and values to alter the appearance of the frame:

```
framem('FLineWidth', 4, 'FEdgeColor', 'red')
```

The Map Grid

In The Mapping Toolbox, the *map grid* is the set of displayed meridians and parallels. Display the grid by setting the map axes property `Grid` to `'on'`. This can be accomplished upon map axes creation with `axesm`, with `setm`, or with the direct command `gridm on`.

To control display of meridians and parallels, set a scalar meridian spacing or a vector of desired meridians in the `MLineLocation` property. The corresponding property `PLineLocation` serves the same purpose for parallels. The default values are every 30° for meridians and every 15° for parallels.

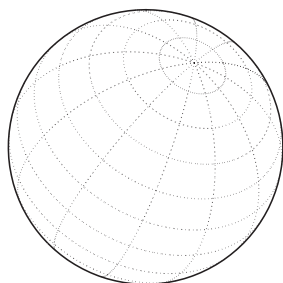


Default Grid on a Miller Projection

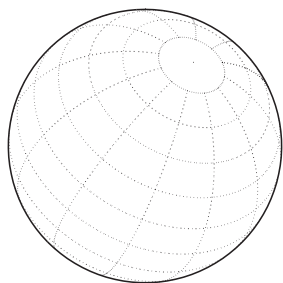
By default, the grid is placed as the top layer of any display. You can alter this by changing the `GAItitude` property, so that other map objects can be placed “above” the grid. The new grid is drawn at its new altitude.

To reposition the grid back to the top of the display, use the command `gridm reset`. You can also control the appearance of grid lines with the `GLineStyle` and `GLineWidth` properties, which are `:` and `0.5`, respectively, by default.

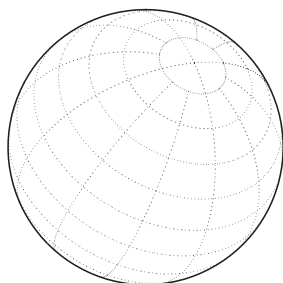
The Miller projection is an example in which all the meridians can extend to the poles without appearing to be cluttered. Look, however, at the following Orthographic projections. When all the meridians extend to the poles, the look is “congested.” The `MLineLimit` property allows you to specify a pair of latitudes at which to terminate the meridians. For example, setting `MLineLimit` to `[-75 75]` completely clears the polar region of displayed meridians. Sometimes you might like to allow some of the meridians to ignore the limit and extend to the pole. If, for example, `MLineException` is set to `[-90 0 90 180]`, then the meridians corresponding to those four longitudes will continue past the limit on to the pole:



Default grid allows all displayed meridians to extend to the poles.



The property `MLineLimit` can truncate meridians at a given latitude, here at 75°N and S.



The property `MLineException` allows certain meridians to extend to the poles despite the `MLineLimit`. Here, four meridians 90°W, 0°, 90°E, and 180° are excepted.

There are also corresponding map axes properties for controlling the extent of displayed parallels: `PLineLimit` and `PLineException`.

Users can label displayed parallels and meridians. `MeridianLabel` and `ParallelLabel` are on-off properties for the display of labels on the meridians and parallels, respectively. They are both 'off' by default. Initially, the label locations coincide with the default displayed grid lines, but you can alter this by using the `PLabelLocation` and `MLabelLocation` properties. These grid lines are labeled across the north edge of the map for meridians and along the west edge of the map for parallels. However, the property `MLabelParallel` allows you to specify 'north', 'south', 'equator', or a specific latitude at which the meridian labels will be displayed, and the `PLabelMeridian` allows the choice of 'west', 'east', 'prime', or a specific longitude for the parallel labels. Refer to the `axesm` reference page in the *Mapping Toolbox Reference Guide* for complete descriptions of all map axes properties.

Map and Frame Limits

In the Mapping Toolbox, the map and frame limits are two related map axes properties that limit the map display to a defined region. The map latitude and longitude limits restrict the displayed parallels and meridians, while the frame limits control the extent of the frame around the displayed data. Any object that extends outside the frame limits is automatically trimmed.

The frame limits are also specified differently from the map limits. The map limits are in Greenwich coordinates referenced to an origin at the intersection of the Prime Meridian and the Equator, while the frame limits are in coordinates referenced to the center of the frame, which is the latitude and longitude of the map axes origin.

For all non-azimuthal projections, frame limits are specified as quadrangles (`[latmin latmax]` and `[longmin longmax]`) in the frame coordinate system. In the case of azimuthal projections, the frames are circular and are described by a polar coordinate system. One of the frame latitude limits must be a negative infinity (-Inf) to indicate an azimuthal frame (think of this as the center of the circle), while the other limit determines the radius of the circular frame (`rlatmax`). The longitude limits of azimuthal frames are inconsequential, since a full circle is always displayed.

If you are uncertain about the correct format for a particular projection frame limit, you can reset them to the default values using empty matrices.

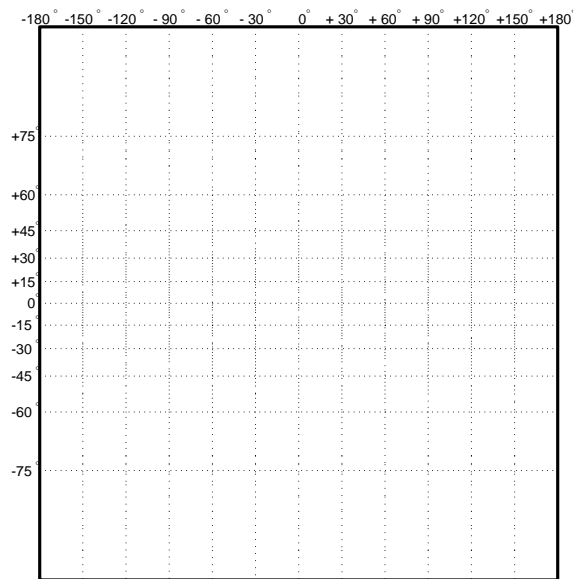
NOTE: For non-azimuthal projections in the normal aspect, the map is limited to the minimum of the map and frame limits; hence, the two limits will coincide after evaluation. When changing one set of limits, you may want to clear the other set to get consistent limits.

Switching Between Projections

When switching between projections using `setm` or the graphical interfaces, you may need to change some of the map axes properties for proper appearance. Settings that are suitable for one projection may not be appropriate for another. Some projections have default properties that define *that* particular projection and may not be altered; for example, the Balthasart Cylindrical projection is defined to have standard parallels (`MapParallel s`) at 50° . Other projections have default properties that are initially set for proper world display; for example, the Mercator projection limits the latitude range to $\pm 86^\circ$ to avoid “blowing up” at the poles.

Although similar projections may share the same set of properties (Miller Cylindrical and Plate Carree Cylindrical), others may be drastically different (Polyconic and Stereographic azimuthal). The classification of the map projections is often a good indicator of whether changes need to be made. For instance, switching from a cylindrical to an azimuthal projection requires a few modifications:

```
axesm mercator
framem on; gridm on; mlabel on; plabel on
setm(gca, 'LabelFormat', 'signed')
```



To display the default map and frame latitude limits for the Mercator projection, type the following:

```
[getm(gca, 'MapLatLi mi t'); getm(gca, 'FLatLi mi t')]  
ans =  
    -86     86  
    -86     86
```

Both the frame and map latitude limits are set to $\pm 86^\circ$ for the Mercator projection to maintain a safe distance from the singularity at the poles.

Now switch the projection to an Orthographic azimuthal:

```
setm(gca, 'MapProj ecti on', 'ortho')
```

What happened to the map frame and labels? If you recall, the frame latitude limits have not been changed and still correspond to the default values for a Mercator projection, as do all the other properties. Only those properties that *must* have particular values are updated for the current projection.

```
getm(gca, 'FLatLi mi t')  
ans =  
    -86     86
```

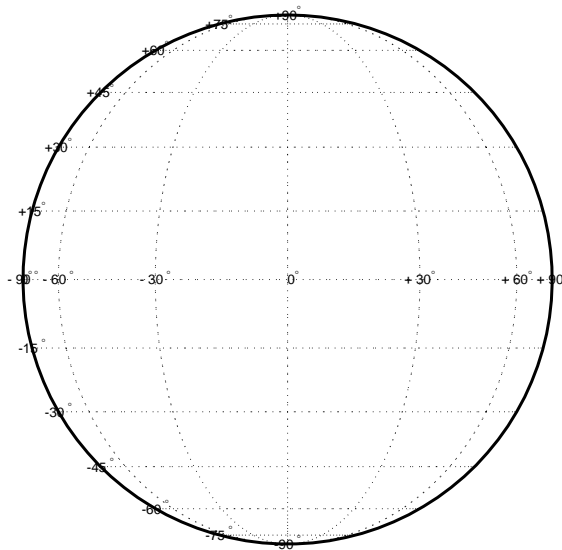
You must manually reset the frame and map limits to appropriate values for an Orthographic projection so that the circular frame is displayed. If you don't know the default or appropriate values, provide an empty matrix for any of the property values:

```
setm(gca, 'FLatLi mi t', [], 'MapLatLi mi t', [])  
[getm(gca, 'MapLatLi mi t'); getm(gca, 'FLatLi mi t')]  
ans =  
    -90     90  
   -Inf     89
```

You will also need to manually specify the locations of the meridian and parallel labels:

```
setm(gca, 'MLabel Paral l el', 0, 'PLabel Meri di an', -90)
```


Now the map is displayed correctly, with the frame:



Default property values can be accessed by the use of empty matrices, as shown in the last example, and the properties can be reset according to the “new” values. The entire set of properties can be reset to their default values by pressing the **Reset** button on the axesmui GUI.

For complete descriptions of all map axes properties, consult the axesm reference page in the “Mapping Reference” section of the *Mapping Toolbox Reference Guide*. For more information on the use of axesmui, refer to its reference page in the “GUI Reference” section of the *Mapping Toolbox Reference Guide*, or read the section of this document entitled “Editing Map Projection and Display Properties” located in Chapter 6, “GUI Tools.”

Projected and Unprojected Objects

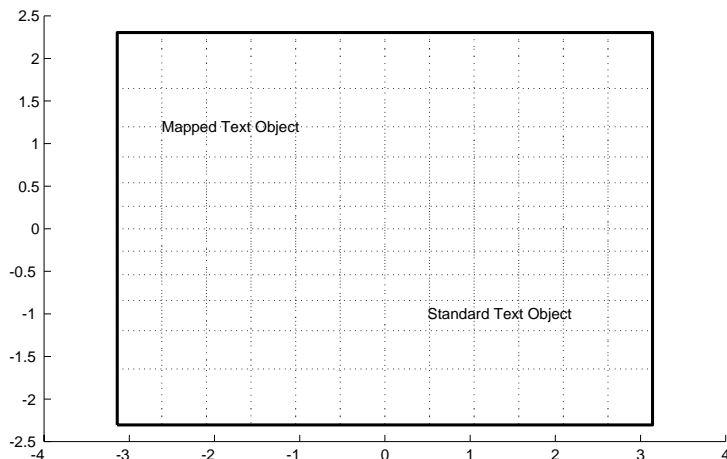
All objects displayed using Mapping Toolbox commands are projected on the map axes based on their designated latitude-longitude positions. The latitudes and longitudes are mathematically transformed to x and y positions using the formulas for the current map projection. Objects displayed using standard MATLAB commands (like `plot`, `fill`, `surface`, and `text`) are placed in the designated x - y location. If the projection changes for a map axes (for example, by using the `setm` command to alter the `MapProjection` property), those objects displayed with standard commands are unaffected, but those displayed using mapping commands are reprojected into new positions. Here is an example:

```
axesm miller; framem on; gridm on;  
showaxes; grid off;
```

This set of commands creates a map axes, a map frame enclosing the region of interest, and geographic grid lines. The x - y axes, which are normally hidden, are displayed, and the MATLAB x - y grid is turned off. It should be noted that the MATLAB x - y grid is separate from the Mapping Toolbox geographic version `gridm`.

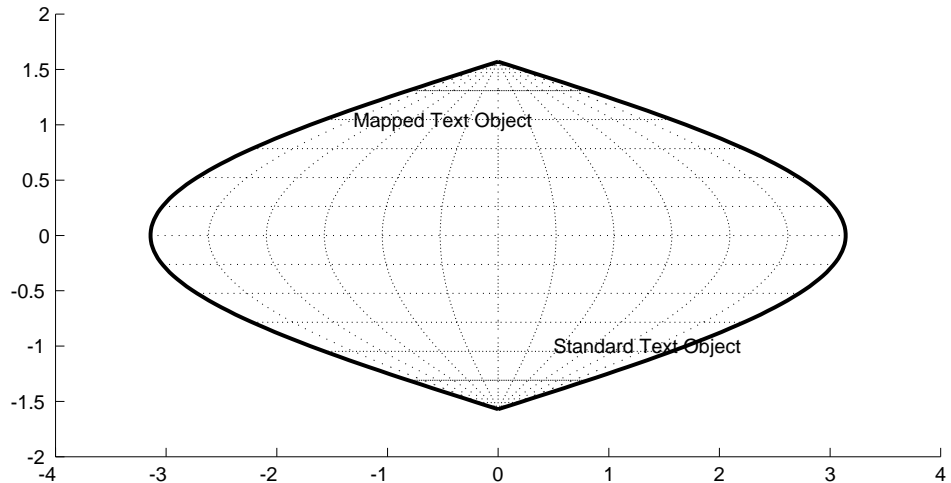
```
text(.5, -1, 'Standard Text Object')  
textm(60, -150, 'Mapped Text Object')
```

Here, a standard text object is placed at $x=0.5$ and $y=-1$, while a mapped text object has been placed at (60°N , 150°W) in the Miller projection.



If we change the projection to a Sinusoidal, the standard text object remains at the same Cartesian position, so its latitude-longitude position is altered. The mapped text object remains at the same geographic location, so its x-y position is altered. Also, the frame and gridlines are replaced according to the new map projection:

```
setm(gca, 'MapProjection', 'sinusoid')  
showaxes; grid off;
```



Similarly, vector and matrix data can be displayed using either mapping or standard commands.

Displaying Vector Maps as Lines

The Mapping Toolbox lets you display vector map data as line objects much like the line display commands in MATLAB. The Mapping Toolbox line graphics functions have MATLAB analogs, the names of which can usually be determined by appending an ‘m’ to the MATLAB function name. For instance, the Mapping Toolbox version of `plot` is `plotm`.

The following table lists the available Mapping Toolbox line display functions:

Function	Used to Create
<code>contorm</code>	Contour plot of map data
<code>contor3m</code>	Contour plot of map data in 3-D space
<code>linem</code>	Line objects projected on map axes
<code>plotm</code>	Lines projected on map axes
<code>plot3m</code>	Lines projected on map axes in 3-D space

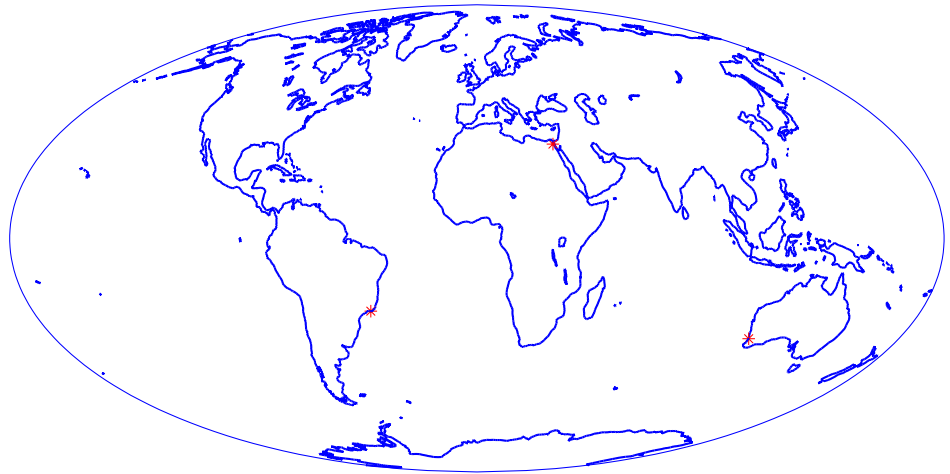
Plot the coast vector data using `plotm`. Just as with `plot`, you can specify line property names and values in the command:

```
load coast
axesm mollweid
framem('FEdgeColor','blue','FLineWidth',0.5)
plotm(lat,long,'LineWidth',1,'Color','blue')
```

Sometimes vector data represent specific points. Suppose you have variables representing the locations of Cairo (30°N,32°E), Rio de Janeiro (23°S,43°W), and Perth (32°S,116°E), and you want to plot them as markers only, without connecting line segments.

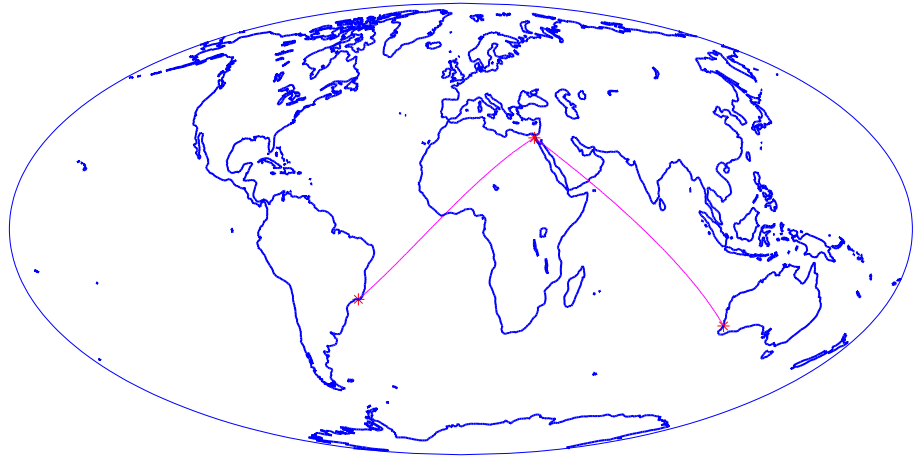
Here is the map:

```
citylats = [30 -23 -32]; citylongs = [32 -43 116];  
plotm(citylats, citylongs, 'r*')
```



In addition to these sorts of “permanent” geographic data, you can also display calculated vector data. Calculate and plot a great circle track from Cairo to Rio de Janeiro, and a rhumb line track from Cairo to Perth:

```
[gcl at, gcl ong] = track2('gc', ci tyl ats(1), ci tyl ongs(1), ...  
                           ci tyl ats(2), ci tyl ongs(2));  
[rhl at, rhl ong] = track2('rh', ci tyl ats(1), ci tyl ongs(1), ...  
                           ci tyl ats(3), ci tyl ongs(3));  
plotm(gcl at, gcl ong, 'm-'); plotm(rhl at, rhl ong, 'm-')
```



Displaying Vector Maps as Patches

Vector map data that is properly formatted can be displayed as a patch, or filled-in polygon. Like many Mapping Toolbox functions, the names of the patch map display functions can be determined by appending an 'm' to its MATLAB counterpart (e.g., add an 'm' to patch to get patchm).

NOTE: The Mapping Toolbox patch display commands differ from their MATLAB equivalents by allowing you to display patch vector data that use NaNs to separate faces.

The following table lists the available Mapping Toolbox patch display functions:

Function	Used to Create
fillm	Filled 2-D map polygons
fill3m	Filled 3-D map polygons in 3-D space
patchm	Patch objects projected on map axes
patchesm	Patches projected as individual objects on map axes

The usalo MAT-file is useful for illustrating patches:

```
load usalo
who
Your variables are:
```

conus	gtlakeon	statelat	uslon
greatlakes	state	statelon	
gtlakeat	stateborder	uslat	

The variables `uslat` and `uslon` together describe three polygons (separated by NaNs) representing the continental United States. The smaller polygons are necessary to display Long Island and Martha's Vineyard. The variables `gtlakelat` and `gtlakelon` describe three polygons (separated by NaNs) for the Great Lakes. The variables `statelat` and `statelon` contain line-segment data (separated by NaNs) for the borders between states, which is not formatted for patch display.

Conic projections make good displays of the United States, so create a map axes using an Albers Equal-Area Conic projection. By specifying map limits that contain the region of interest, you automatically center the projection on an appropriate longitude, and the frame encloses just the mapping area, not the entire globe.

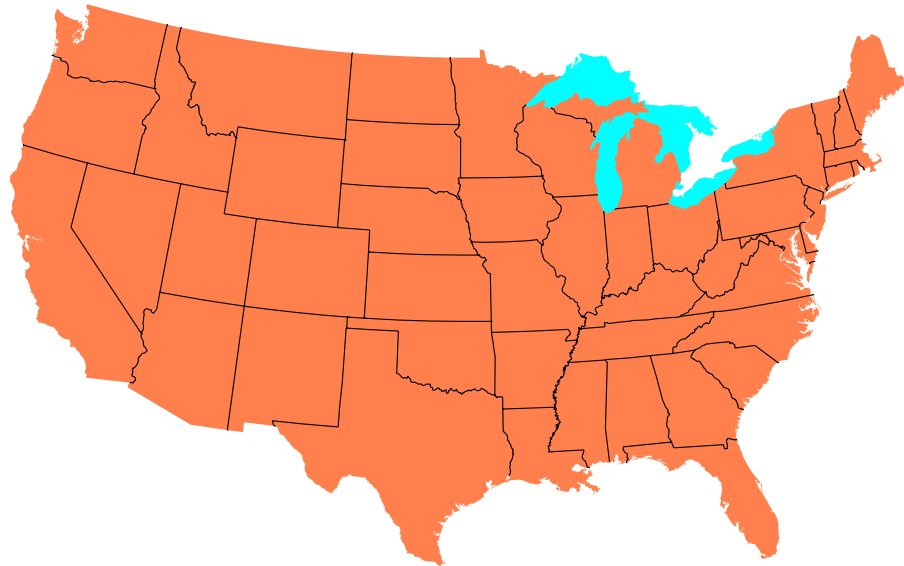
A good rule-of-thumb for conic projections is to set the standard parallels at one-sixth of the distance from both latitude limits. You can do this automatically by providing an empty matrix as the standard parallel:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[23 52],'MapLonLimit',[-130 -62])
```

When patch data is displayed, layering can become important as objects above can hide objects below. You can control the visibility of objects by the order of their display and by their “altitude” position. For this reason, the primary patch display function in the Mapping Toolbox, `fillm`, allows you to control the z-axis level of displayed patches. Here the land data is displayed at the default level, `z=0`. The Great Lakes data is assigned the altitude of `z=1`. To plot the line segment data between these layers, use the `plot3m` command and specify an altitude of 0.5:

```
fillm(uslat,uslon,'FaceColor',[1 .5 .3],'EdgeColor','none')
fillm(gtlakelat,gtlakelon,1,...
      'FaceColor','cyan','EdgeColor','none')
plot3m(statelat,statelon,0.5,'k')
```


Here is the result:



The `fillm` function makes use of the low-level function `patchm`. The Mapping Toolbox provides another patch drawing function called `patchesm`. The optimal use of either depends on the application and user preferences. The `patchm` function creates one displayed object and returns one handle for a patch, which may contain multiple, not necessarily connecting, faces. The Mapping Toolbox uses NaNs to separate non-connected patch faces, unlike MATLAB, which does not handle NaN clipped data for patches. For the `patchesm` function, each face is a separate object with its own handle. In general, `patchm` requires more memory but is faster than `patchesm`.

Displaying Matrix Maps

The Mapping Toolbox provides functions for the display and enhancement of both regular and general matrix maps in a variety of formats. Recall that regular matrix maps require a map legend vector that describes the scale and location of the map, while general matrix maps require matrices of latitude and longitude coordinates.

The matrix map display functions are geographic analogies to the MATLAB surface drawing commands. Like the line plotting functions discussed in the previous chapter, the Mapping Toolbox function names are usually identical to their MATLAB counterparts, except for the addition of an 'm.' For instance, the Mapping Toolbox functions, `surfacem` and `surfm`, are analogous to MATLAB's `surface` and `surf` commands. You will most likely find a Mapping Toolbox version of a MATLAB surface drawing function by appending an 'm' to the MATLAB function name.

NOTE: In the Mapping Toolbox, the set of functions beginning with 'mesh' are used for regular matrix maps, while those with 'surf' are reserved for general matrix maps. It should be noted here that this usage differs entirely from MATLAB's definition; that is, `mesh` plots are used for colored, wire-frame views of the surface, while `surf` displays colored, faceted surfaces.

Surface map objects can be displayed in a variety of different ways. You can assign colors from the figure colormap to surfaces according to the value of their data. You can also display images where the matrix data consists of indices into a colormap or display the matrix as a three-dimensional surface, with the z coordinates given by the map matrix. You can use monochrome surfaces that reflect a "pseudo-light" source, thereby producing a three-dimensional, shaded relief model of the surface. Finally, you can use a combination of color and light shading to create a lighted shaded relief map.

The following table lists the available Mapping Toolbox surface map display functions:

Function	Used to Create
<code>meshm</code>	Regular matrix map warped to projected graticule mesh
<code>surf</code>	General matrix map projected on map axes
<code>image</code>	Regular matrix map displayed as image
<code>pcolorm</code>	Projected matrix map in $z = 0$ plane
<code>surfacem</code>	Matrix map warped to projected graticule mesh
<code>surf</code>	3-D shaded surface with lighting projected on map axes
<code>meshl</code>	3-D lighted shaded relief of regular matrix map
<code>surf</code>	3-D lighted shaded relief of general matrix map

The Graticule

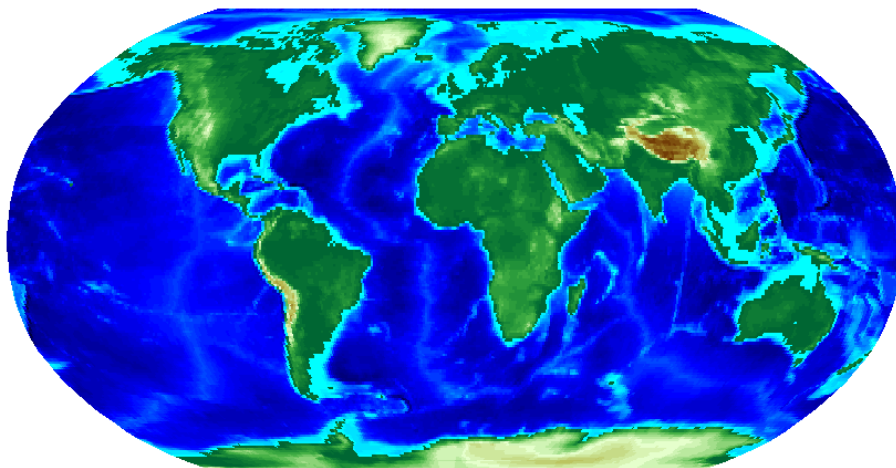
The Mapping Toolbox projects surface objects in a manner very similar to the traditional methods of map making. The cartographer traditionally lays out a grid of meridians and parallels, called the *graticule*. Each graticule cell is therefore a geographic quadrangle. The cartographer calculates the appropriate x - y locations for every vertex in the graticule grid and draws the projected graticule by “connecting the dots.” Finally, the cartographer draws the map data in “freehand,” attempting to account for the shape of the graticule cells. Similarly, the Mapping Toolbox calculates the x - y locations of the four vertices of each graticule cell and warps the matrix data to fit the resulting quadrilateral.

In the toolbox, as in traditional cartography, the finer the graticule mesh (i.e., the more meridians and parallels used), the greater the precision of the projected map display, at the cost of greater effort and time. Graticules for regular matrix maps are defined in the toolbox as a two-element vector, of the form `[number-of-parallels, number-of-meridians]`. For general matrix maps, the graticule is related to the size of the latitude and longitude coordinate matrices, where `number-of-parallels=mrows-1` and `number-of-meridians=ncols-1`.

While the graticule cell for a regular matrix map is restricted to equal-angle quadrangles (i.e., length of cell in latitude must equal length of cell in longitude), general matrix maps have no such constraint. Their graticule cells can be of any size.

The topo regular matrix map can be displayed quickly using a coarse graticule. The cost is in precision:

```
load topo
figure; axesm robinson
graticule = [10 20];
h = meshm(topo, topolegend, graticule);
demcmap(topo)
```

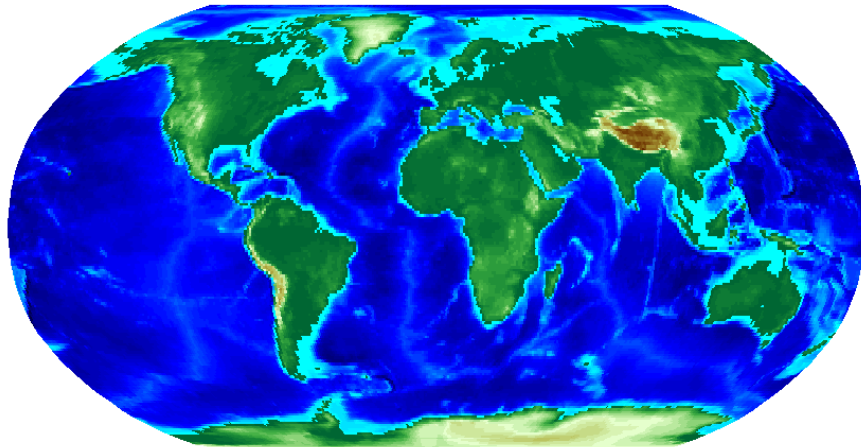


Notice that for this coarse graticule, the edges of the map do not appear as smooth curves. What may not be as obvious is that the easternmost column of graticule cells and the southwesternmost cell are sometimes invisible on displayed matrix maps. This is necessary for the proper projection of the surface object and is not a concern except with the coarsest graticules. Previous displays used the default [50 100] graticule, for which this effect is negligible.

Regardless of the graticule resolution, the matrix map data is unchanged. In this case, the matrix map is the 180-by-360 topo matrix, and regardless of projection fidelity, the resolution of its value data is unchanged.

Map objects displayed as surfaces have all the properties of any MATLAB surface, which can be set at object creation or by using the MATLAB `set` command. The mapping `setm` command allows the `MeshGrat` graticule property to be manipulated for regular matrix surfaces. Since you saved the handle of the last displayed map, reset its graticule to a very fine grid. Remember, the trade-off is between resolution and time, so this could take a while:

```
setm(h, 'MeshGrat', [200 400])
```



You'll probably notice that the result does not appear to be any better than the original display with the default [50 100] graticule, but it took much longer to produce. Actually, the 200-by-400 graticule grid is finer than the 180-by-360 data resolution. There is really no reason to ever use a graticule finer than your data. In practice, you will probably find that coarse graticules are good for development tasks and fine graticules serve well for final graphics production.

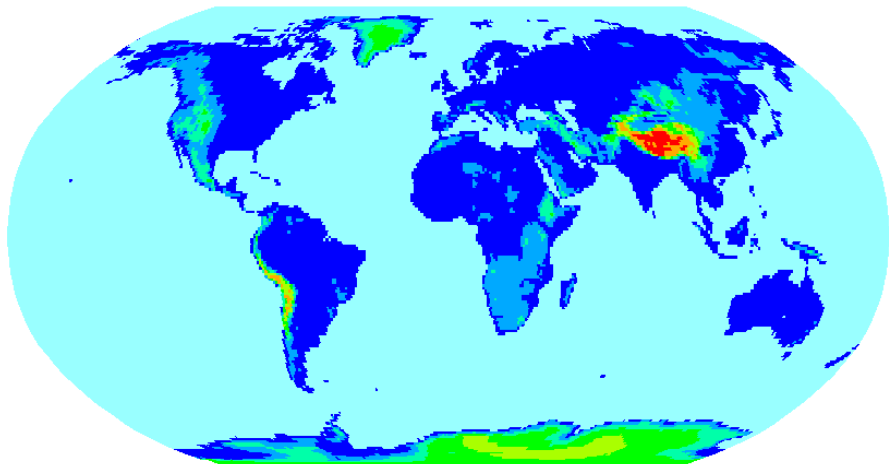
Coloring Matrix Maps

In this tutorial, you may have noticed the use of the function `demcmap` in several digital elevation map (DEM), or topographic, display examples. This function creates colormaps appropriate for the display of DEMs, although it is certainly not limited to just DEMs.

The colormaps, by default, have atlas-like colors varying with elevation or depth that properly preserve the land-sea interface. The `demcmap` function can also be used to create colormaps very different from the default atlas colors.

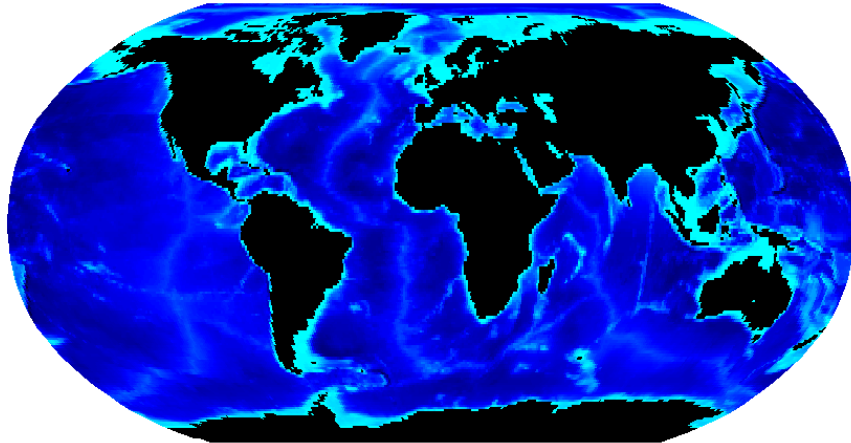
Suppose you want to display the topo map using 16 total colors, coloring the oceans a uniform light-cyan and the land similar to MATLAB's `hsv` colormap:

```
load topo
landclr = [0 0 1; 0 1 0; 1 0 0];
axesm robinson
meshm(topo, topol legend)
demcmap(topo, 16, [.7 1 1], landclr)
```



Instead of specifying the size of the colormap, you can make all values within an elevation band the same color. Modify the colormap to use a surface data interval of 100, coloring all land areas black, and using the default sea colors to take a closer look at the bathymetry data:

```
demcmmap('inc', topo, 100, [], [0 0 0])
```



Use the string 'inc' as the first input argument to indicate that a desired elevation interval or increment is to be used to build the colormap. You can also employ the default colors for the sea by entering an empty matrix in place of an RGB color vector.

Data Representation

Image and Surface Coloring

The simplest way to display a matrix map is to assign colors to matrix elements according to the value of their data. The map can be displayed either as a 2-D image or a 3-D surface using the map values as the z data.

Consider the matrix map located in the korea workspace. It contains a matrix of land elevations and bathymetry data for the region around the Korean peninsula, along with a map legend variable, which indicates a regular matrix map format. Calculate a graticule for the regular matrix map using the `meshgrat` function so that a general matrix map is also available. Also, convert the units for the map matrix from meters to degrees, so they are consistent with the latitude and longitude coordinate matrices.

```
load korea
[lat, lon] = meshgrat(map, maplegend);
map = km2deg(map/1000);
whos
```

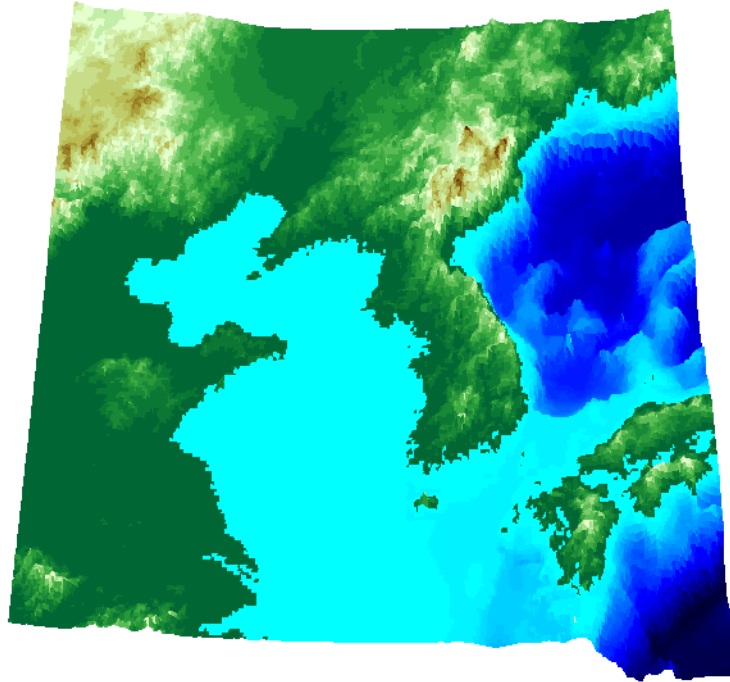
Name	Size	Bytes	Class
lat	180x240	345600	double array
lon	180x240	345600	double array
map	180x240	345600	double array
maplegend	1x3	24	double array

Notice that the `lat` and `lon` coordinate matrices, or graticule, are the same size as the map matrix, a requirement for constructing 3-D surfaces.

You have already seen the `meshm` command used to display regular matrix maps, so display the korea matrix map using the `surf` command:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[30 45],'MapLonLimit',[115 135])
surf(lat,lon,map,map); demcmap(map)
set(gca,'DataAspectRatio',[1 1 .5])
view(0,75)
```


Here is the result:

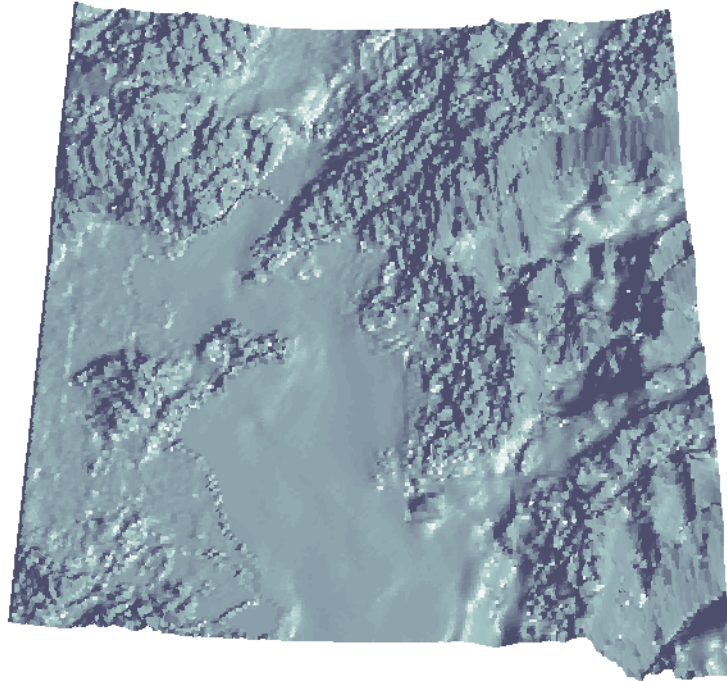


Surface Light Shading

A monochrome three-dimensional, shaded-relief map can be displayed with the function `surf1m`, which is analogous to the MATLAB `surf1` command. This functionality is available only for general matrix maps. Remember that regular matrix maps are a subset of general matrix maps and can be easily converted.

To display a map in this format, enter the following:

```
cl ma  
surf1 m(lat, lon, map); colormap(bone)
```



This is the same region as the previous map, viewed in three dimensions. Notice the dramatic dip in the lower-right portion of the map caused by the deep ocean trenches of the North Pacific Ocean. The view is from the south, at an elevation angle of 75° . The light source in this case is the default 45° counterclockwise from the view direction.

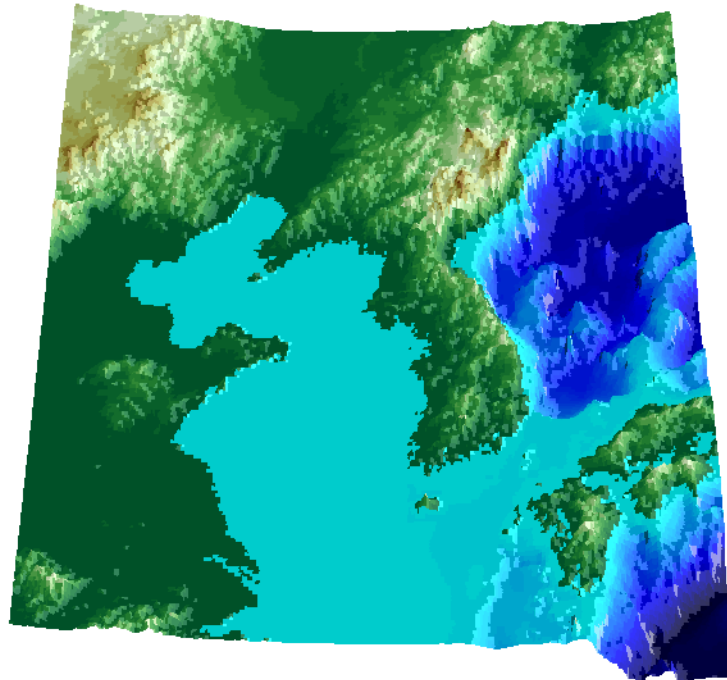
This kind of representation shows much more of the fine structure of the land and sea floor, but because of the lack of color, it is difficult to distinguish land from sea.

Surface Lighted Shaded Relief

The commands `meshl srm` and `surf1 srm` display maps in a lighted shaded relief format and can be thought of as extensions to `surf1 m` that combine surface coloring and surface light shading. Again, `meshl srm` is used for regular matrix maps and `surf1 srm` for general matrix maps. Although there are no analogous MATLAB functions, you can obtain similar results using MATLAB light objects. However, unlike MATLAB lighted surfaces that do not modify the surface's `CData`, these mapping commands construct a new colormap and associated `CData` matrix by using grayscales to lighten or darken a matrix component based on its calculated surface normal to a light source.

To display a lighted shaded relief map, type the following:

```
cl ma  
surf1 srm(lat, lon, map)
```

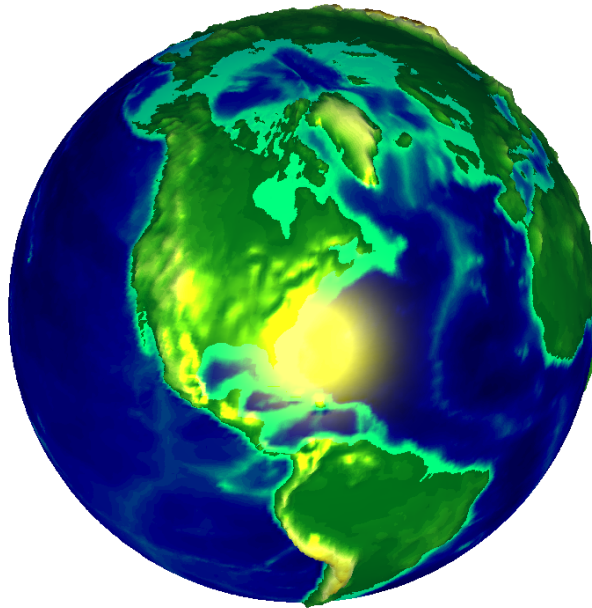


Mapped Light Objects

The Mapping Toolbox allows for the mapping of light objects using the `lightm` command, which is similar to the `light` command in MATLAB. Here, place a lighted source at an infinite distance, representing the Sun at the summer solstice (23.5°N) at local apparent noon in Natick, Massachusetts (71°W):

```
load topo
ptopo = topo; ptopo(topo<0) = 0;
ptopo = 100*ptopo/(almanac('earth','radius')*1000);

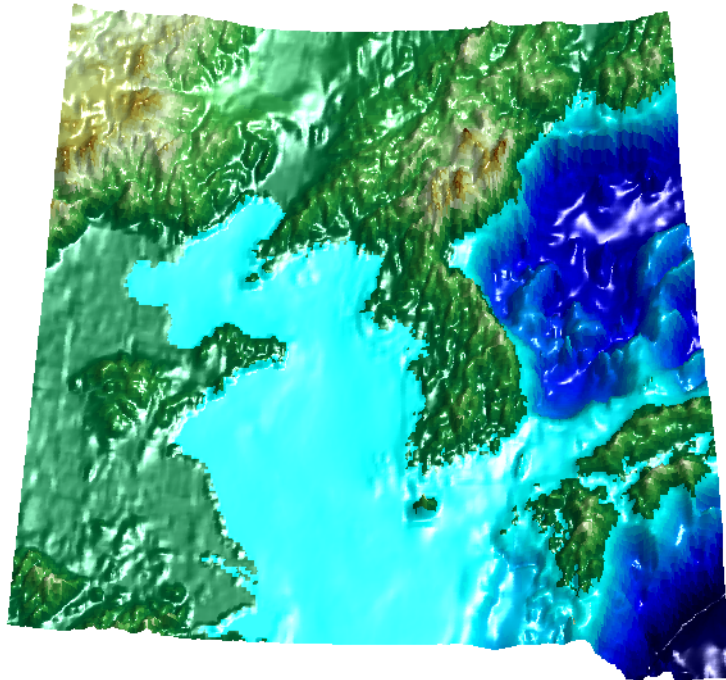
axesm globe; view([15 38])
meshm(topo, topolegend, size(topo), ptopo); demcmap(topo)
shading interp
lightm(23.5, -71, 'Color', 'y')
material([.5 .7 1.5]); lighting phong
```



The topo surface had been modified to show the 3-D features of land only, which is vertically exaggerated by a factor of 100. The ocean depths are shown by color only.

As a comparison to the lighted shaded relief example shown earlier, add a light source to the surface colored matrix map of the Korean peninsula region shown earlier:

```
figure; load korea
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[30 45],'MapLonLimit',[115 135])
meshm(map,maplegend,size(map),map); demcmap(map)
lightm(37.5,125,'Position',[1 0 1])
material([.5 .5 1]); lighting phong
set(gca,'DataAspectRatio',[1 1 50000])
view(0,75)
```



The light effects employed here are different and separate from those used by `surf`, `meshl srm`, and `surfl srm`. Here, a light object is added directly to the map surface. Like all displayed surfaces, the light object is a separate MATLAB graphic object with its own object handle. For the functions `meshl srm` and `surfl srm`, light effects are simulated within the functions by modifying the colormap with bands of light and dark. The map matrix is then converted to indices for the new “shaded” colormap based on calculated surface normals.

Using light objects allows for a wide range of different light effects, such as ambient light, light color, and surface reflectance properties, but will ultimately require more memory and time to display.

The `meshl srm` and `surfl srm` functions provide a simple means of displaying maps with simulated light effects. Both shaded relief formats represent the best available method of displaying matrix maps, since both large (surface coloring) and small (surface shading) features can be examined on one map.

For more information, consult the reference pages for `meshl srm`, `surfl srm`, and `lightm` in the *Mapping Toolbox Reference Guide*, along with the section on lighting in the MATLAB document, *Using MATLAB Graphics*.

The Geographic Data Structure

In the previous examples, all of the map data was in the form of individual variables and had to be displayed using different mapping commands according to the type of data at hand (i.e., line, patch, matrix, text, etc.). The Mapping Toolbox also provides an easy means of displaying, extracting, and manipulating collections of all types of map objects that have been organized in a specially defined and formatted *geographic data structure*. Note that this structure is different from the *map projection structure*, which defines a map projection along with its mapping properties. The geographic data structure contains the information required for the proper display of the graphic object. The following table lists the six object types along with their required fields of information:

fields	light	line	patch	regular	surface	text
type	•	•	•	•	•	•
tag	•	•	•	•	•	•
lat	•	•	•		•	•
long	•	•	•		•	•
map				•	•	
maplegend				•		
meshgrid				•		
string						•
altitude	•	•	•	•	•	•
otherproperty	•	•	•	•	•	•

Some fields may contain empty entries, but the fields themselves must exist for the object to be displayed correctly. For instance, the `altitude` field can be an empty matrix and `otherproperty` can be an empty cell array. The `tag` field must be a string not equal to the name of the object type (i.e., `line`, `surface`, `text`, etc.). For complete descriptions of these fields and the object types, consult the geographic data structure reference page in the *Mapping Toolbox Reference Guide*.

The `usal` workspace contains several variables in the geographic data structure format:

```
load usalo
conus
conus =
    lat: [4392x1 double]
    long: [4392x1 double]
    type: 'patch'
    tag: 'ContinentalUnitedStates'
    otherproperty: []
    altitude: []

stateborder
stateborder =
    lat: [2345x1 double]
    long: [2345x1 double]
    type: 'line'
    tag: 'StateBorder'
    otherproperty: { 1x1 cell }
    altitude: []

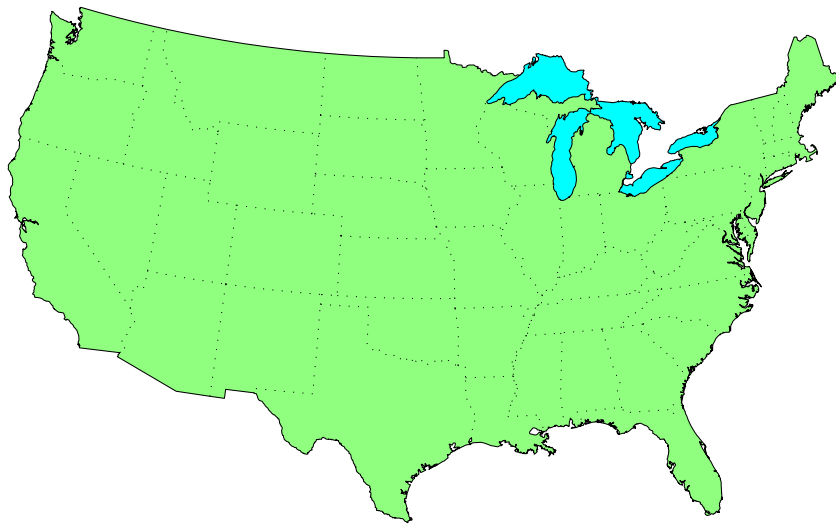
greatlakes
greatlakes =
1x3 struct array with fields:
    type
    tag
    lat
    long
    altitude
    otherproperty
```


The structure `conus` contains a patch object of the continental U.S.; `greatlakes` consists of three patches representing the Great Lakes, and the structure `stateborder` contains line vector data for the state boundaries.

These can be displayed with the `displaym` function:

```
axesm('MapProjection','lambert','MapParallels',[],...
      'MapLatLimit',[23 52],'MapLonLimit',[-130 -62])
displaym(conus)
displaym(greatlakes)
displaym(stateborder)
```

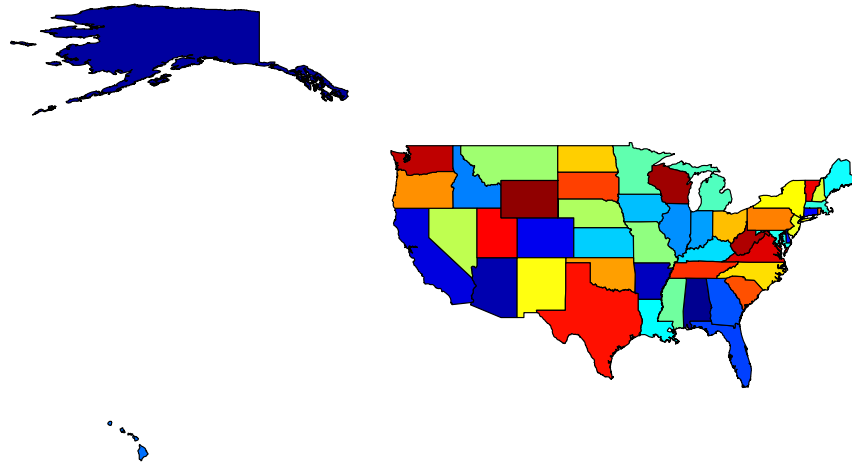
The map shown here is virtually identical to that of a previous example in which the vector variables `uslat`, `uslon`, `gtlakeat`, `gtlakeon`, `stateat`, and `stateon` were used:



Now display all 50 states and the District of Columbia using the patches located in the geographic data structure variable `state`:

```
figure
axesm trystan
displaym(state)
```

Here is the map in a Trystan Edwards Cylindrical projection:



You can also treat geographic data structures as databases of map data. The `extractm` function lets you retrieve data from the formatted structure. For example, you can extract the state of Massachusetts from the `state` structure:

```
[lat, long] = extractm(state, 'mass');
```

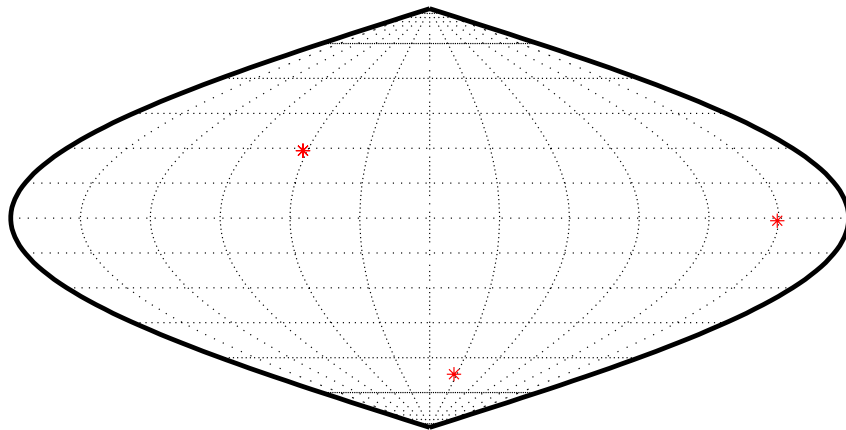
NOTE: Most of the Mapping Toolbox atlas data and data from public sources processed by Mapping Toolbox external interface functions enters the MATLAB workspace as geographic data structures. You can also format your own data as geographic data structures.

Interacting with Displayed Maps

The Mapping Toolbox provides the means to interact with displayed maps. The `inputm` command allows you to get the latitude-longitude position of a mouse-click; the `gcpmap` function returns the current mouse position, again in latitude and longitude. As an example, display a map axes with its grid. The lines of code below allow you to click on the map three times and store the geographic coordinates for the three points selected in `lat` and `long`. The final `plotm` command will plot the points you clicked on. The display shown here corresponds to the points chosen for the example:

```
axesm sinusoid
framem on; gridm on
points=inputm(3)
points =
    28.9373  -62.1631
    -1.1575  149.3467
    -67.1344  26.8096

plotm(points, 'r*')
```



Your points will probably be different. You can also use special interactive commands to interactively place text, small circles, and tracks. For more information on these capabilities, consult the `gtextm`, `scircleg`, and `trackg` entries in the *Mapping Toolbox Reference Guide*, along with Chapter 6 of this manual, “GUI Tools.”

The Mapping Toolbox also allows you to manipulate displayed objects by name. Many mapping commands assign descriptive names to the Tag property of the objects they create. The name functions allow you to control the display of groups of similarly named objects, determine the names and change them if so desired, and use the name in the Handle Graphics set and get commands. There is also a Mapping Toolbox graphical user interface, `mobjects`, to help you manage the display and control of objects.

Some mapping display functions like `framem`, `gridm`, and `contorm`, assign object tags by default. You can also set the name upon display by assigning a string to the Tag property in mapping display functions that allow property-value pairs. If the Tag does not contain a string, the name falls back to an object's Type property, such as 'line' or 'text'.

Display a vector map of the world:

```
figure; axesm fournier
framem on; gridm on;
plabel on; mlabel('MLabel Parallel', 0)

load coast
plotm(lat, long, 'k', 'Tag', 'Coastline')
```

You can view the names of the objects in the current axes using `namem`:

```
namem
ans =

Coastline
PLabel
MLabel
Meridian
Parallel
Frame
```

Change the line width of the coastline:

```
set(handles('Coastline'), 'LineWidth', 2)
```

Hide, then show all of the objects of type 'text', regardless of their tags:

```
hide('alltext')
show('alltext')
```

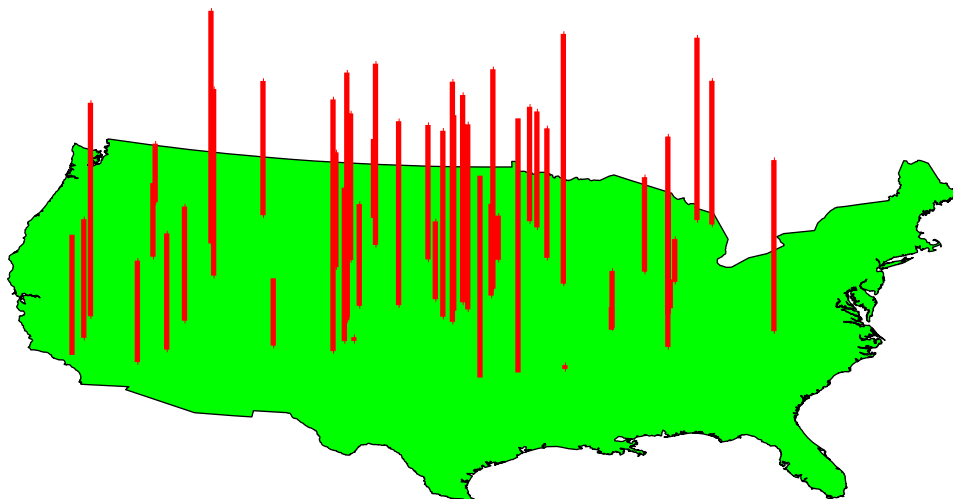
For more information on the use of these functions and tools, consult the *Mapping Toolbox Reference Guide* under the headings for `handles`, `hide`, `show`, `clm`, `namem`, `tagm`, and `objects`.

Specialized Map Displays

Thematic Map Functions

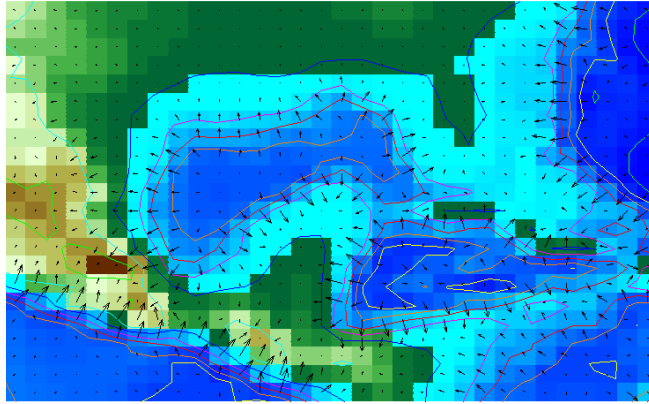
The Mapping Toolbox provides a wide variety of other display and symbology functions. The `cometm` and `quiverm` functions operate like their MATLAB counterparts, `comet` and `quiver`. The `stem3m` command allows you to display geographic bar graphs. Like the MATLAB `scatter` command, the `scatterm` function allows you to display a thematic map with proportionally sized symbols. The `tissot` function calculates and displays Tissot Indicatrices. For more information on these capabilities, consult the descriptions of these functions in the “Mapping Reference” section of the *Mapping Toolbox Reference Guide*.

Here is an example of a stem plot over a map of the continental United States. The bars could represent anything from selected city populations to the number of units of a product purchased at each location:

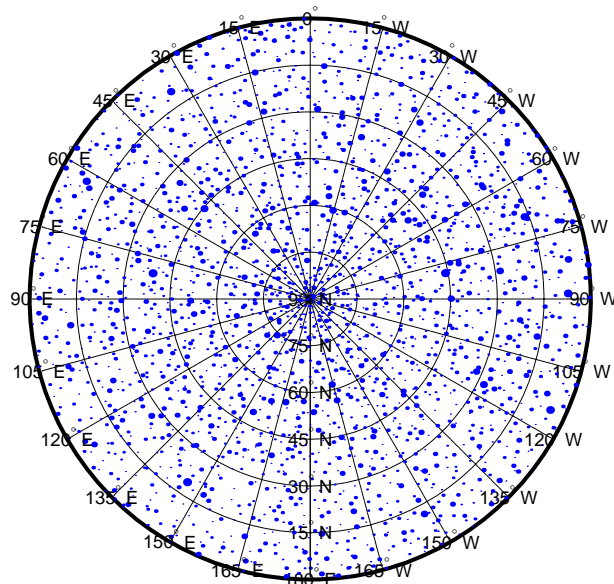


Contour and quiver plots can be extremely useful in analyzing matrix data. In the following example, contour elevation lines have been drawn over a topographical map. The region displayed is the Gulf of Mexico, obtained from the topo matrix. Quiver plots have been added to visualize the gradient of the topographical matrix.

Here is the displayed map:



Here is an example of using the `scat term` function to create a star chart of the northern sky. The stars are represented by filled circles whose size is proportional to visual magnitude. Star gazers may recognize the Big Dipper constellation, or Ursa Major, located at 180° E, 60° N:



Choropleth Maps

A common type of map display has patch faces colored proportional to some specified data. The data generally represents the value of something per unit area, such as population density, electoral votes, or incidence of disease. Maps of this type are generally called *choropleth* maps. It is easy to construct choropleth maps with MATLAB and the Mapping Toolbox. Simply assign the data values to the `CData` property of the displayed patches.

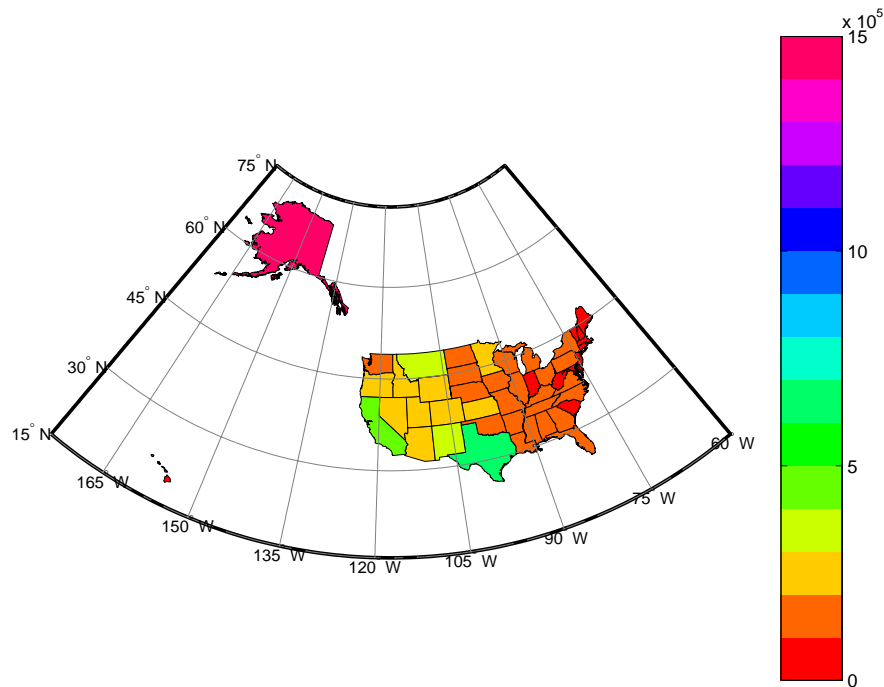
In the following example, patches representing the 50 states of the U.S. (and the District of Columbia) are displayed and colored according to the surface areas calculated by the `areaInt` function. An equal-area projection is appropriate for this kind of display.

```
load usalo
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[15 75],'MapLonLimit',[-175 -60],...
      'MLineLocation',15,'MLabelParallel','south',...
      'MeridianLabel','on','ParallelLabel','on',...
      'GLineStyle','-','GColor',0.5*[1 1 1],...
      'Grid','on','Frame','on')
displaym(state)

geoid = almanac('earth','geoid');
tags = {state.tag};
for i=1:length(state)
    lat = state(i).lat; long = state(i).long;
    surfarea = sum(areaInt(lat,long,geoid));
    set(handles(tags{i}), 'CData', surfarea);
end

axis([0 1.5E6])
colormap(hsv(15))
colorbar
```


The coloring scheme of the map makes it easy to distinguish which states are comparable in area. The color scale to the right of the map shows the surface areas in square kilometers:



The largest state is Alaska (about 1.5 million square kilometers), while Rhode Island stands as the smallest state (about 2500 square km). The District of Columbia, of course, is the smallest entity of the dataset, at roughly 200 square km.

Atlas Data

Types of Data	3-2
World Vector Data	3-3
Coastlines	3-3
World Atlas Data	3-5
World Matrix Data	3-14
Political	3-14
Terrain	3-17
United States Vector Data	3-18
Low Resolution Data	3-18
Medium Resolution State Outlines	3-23
United States Matrix Data	3-26
Political	3-26
Terrain	3-28
Astronomical Data	3-30

Types of Data

The Mapping Toolbox includes a number of datasets for global and regional displays. Map data is available in both vector and matrix format, covering the world and the United States. The Mapping Toolbox also provides astronomical data for simple stellar cartography.

More detailed data is available over the Internet or on CD-ROM and can be imported to MATLAB using the data interface functions described in Chapter 7, “External Data Interface.”

World Vector Data

Coastlines

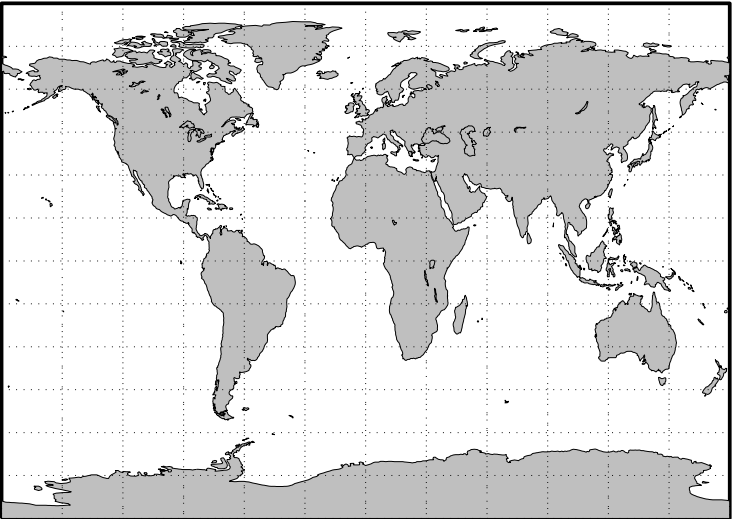
The coast MAT-file contains a set of vector shorelines intended for global displays in which political boundaries are not required. The data is stored in vectors of latitude and longitude, consisting of nearly 10,000 points. While this a considerable quantity of data, it is of very low resolution by cartographic standards.

```
load coast
whos
```

Name	Size	Bytes	Class
lat	9589x1	76712	double array
long	9589x1	76712	double array

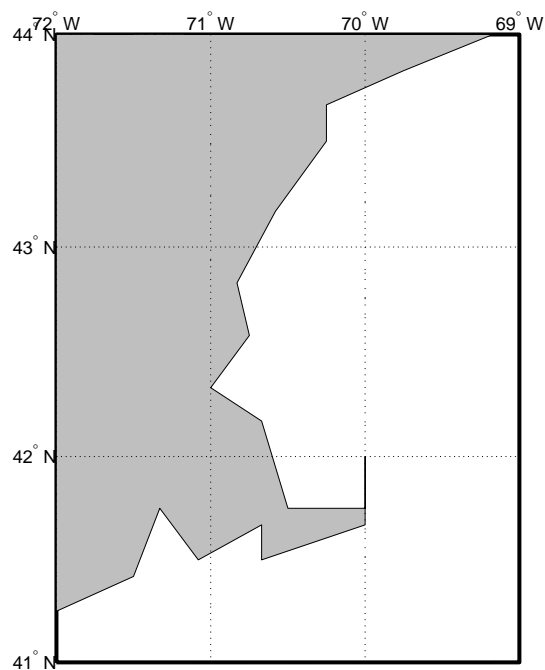
The dataset is in patch format, meaning it has polygon segments that return to their starting points and can therefore be usefully displayed using patch commands. The polygon segments are separated by NaNs into about 240 faces. The data can also be displayed as lines or points.

```
axesm giso; framem; gridm
patchsm(lat, long, 'FaceColor', 0.75*[1 1 1])
```



The resolution for this dataset makes it ideal for global-scale displays but inappropriate for significantly smaller regions, as can be seen when the region around Cape Cod in the northeastern United States is displayed. This is, incidentally, the area covered by the cape MAT-file provided with MATLAB.

```
figure
axesm('MapProjection', 'mercator', ...
      'MapLatLimit', [41 44], 'MapLonLimit', [-72 -69])
framem
gridm('MLineLocation', 1, 'PLineLocation', 1);
mlabel('MLabelLocation', 1)
plabel('PLabelLocation', 1)
patchsm(lat, long, 'FaceColor', .75*[1 1 1])
```



World Atlas Data

The global atlas data contains a set of very small-scale (approximately 1:30,000,000) data for use in global displays. The data is provided in the form of Mapping Toolbox geographic data structures containing national boundaries, rivers, lakes, and cities. Here is a list of the structures in the `worldo` workspace:

```
load worldo
whos
```

Name	Size	Bytes	Class
DNline	1x2	112546	struct array
DNpatch	1x39	59884	struct array
POLine	1x2	456600	struct array
POpatch	1x206	682454	struct array
POtext	1x194	188132	struct array
PPpoint	1x319	400426	struct array
PPtext	1x318	296988	struct array
description	4x75	600	char array
gazette	1x513	353504	struct array

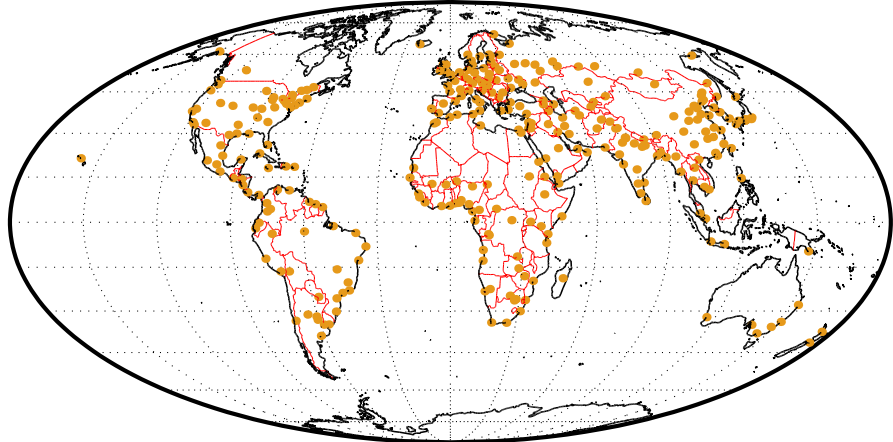
The data has been classified into Drainage (DN), Political/Ocean boundaries (PO), and Populated Places (PP). The data is further separated into structures containing patches, lines, points, or text.

In addition to containing more kinds of data than coast, this dataset is also more detailed. The drainage data consists of about 7,000 points, while the political lines and patches each have about 30,000 points. There are more than 200 political units in the political data and more than 300 major cities in populated places. Because of the quantity of data, complicated displays may require more than MATLAB's default memory size and may take longer to project and render. Displays of patches require the greatest memory and computing time.

The contents of the structures can be displayed from the command line using `displaym`. For example, type the following:

```
axesm mollweid; framem; gridm
displaym(POLine)
displaym(PPpoint)
```

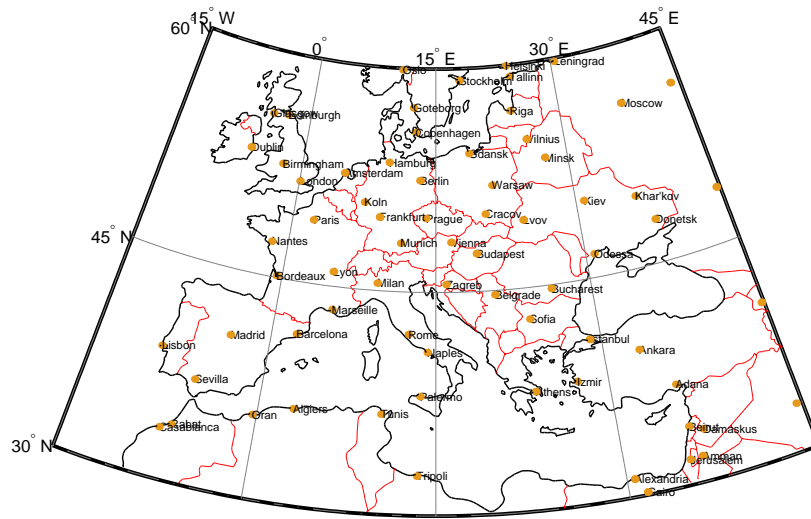
Here is the result:



The map displays the coastlines, international borders, and major cities of the world. We could also label the cities with data from the `PPtext` structure, but at this scale, the result would be unreadable. When we restrict ourselves to a smaller geographic region, more information can be shown:

```
figure
axesm('Mapprojection','eqaconic','MapParallels',[],...
      'MapLatLimit',[30 60],'MapLonLimit',[-15 45],...
      'MLabelLocation',15,'MLineLocation',15,...
      'PLabelLocation',15,'PLineLocation',15,...
      'GColor',.5*[1 1 1],'GLineStyle','-')
framem; gridm; mlabel; plabel
displaym(POLine)
displaym(PPpoint)
displaym(PPtext)
```


The zoomed-in region of Europe allows the names of the major cities to be read more easily:

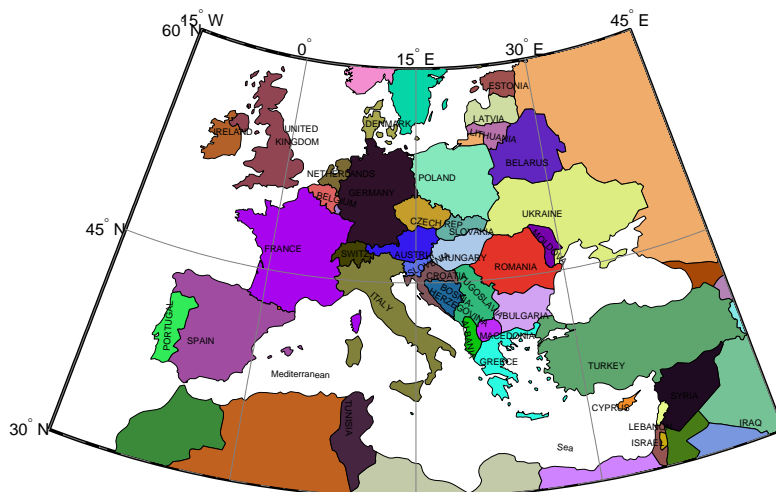


NOTE: Unlike vector and matrix data, text objects are not automatically trimmed when they fall outside the map frame limits. You can remove them later using a graphics program or write a script or function that removes them for you.

There are other ways to represent political data. We can display the countries as patches, label them with strings from the `P0text` structure, and pick random colors for the patch faces. Clear the previous line map, and redraw the patch map:

```
cl ma
di spl aym(P0patch)
di spl aym(P0text)
colormap(rand(200, 3))
```

Here is the result:

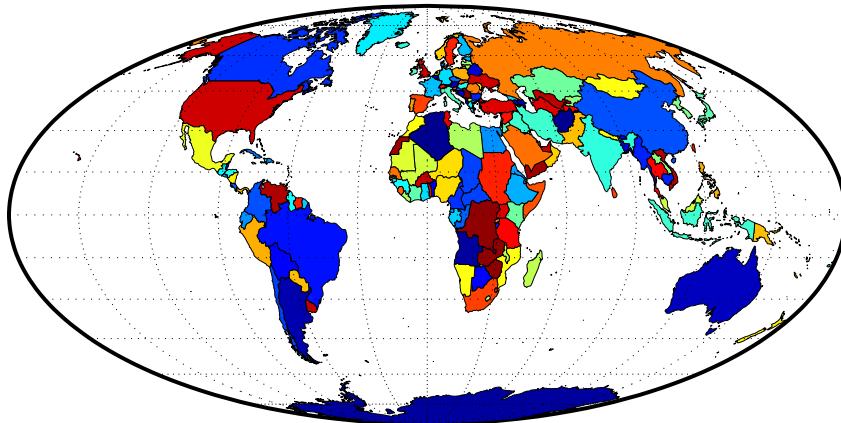


Note how the separated parts of countries are filled with the same colors. Sardinia and Sicily, for example, are shaded the same color as the boot of Italy. This is accomplished automatically because each of the components is tagged as “Italy.”

Every element in a world `dl o` structure has a tag field. Drainage data has been tagged to distinguish between rivers and lakes. The political/ocean lines are tagged as coastlines or international boundaries, and the political patches have been tagged with the names of the country. Both of the functions `di spl aym` and `ml ayes` use the same colors for similarly tagged objects if no other property is specified in the geographic data structure.

In the following example, it is evident that the many islands of Indonesia and Canada are similarly colored, as are all parts of the United States:

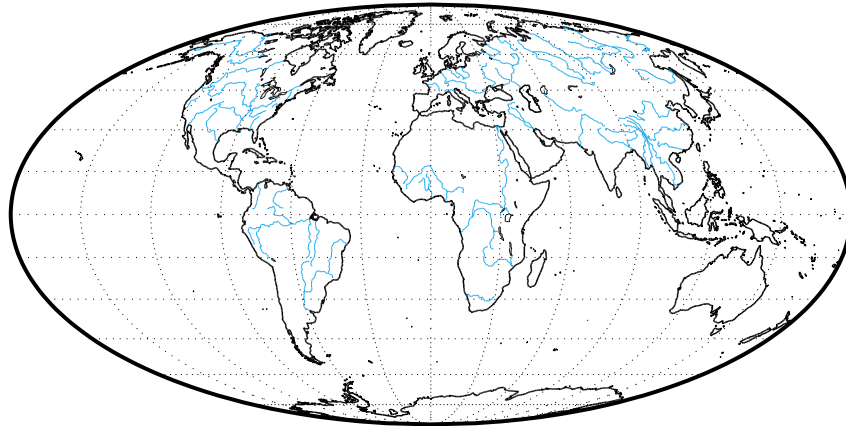
```
figure
axesm mollweid; framem; gridm
displaym(P0patch)
```



You can use the tags to reduce the amount of data in the display by selectively deleting data. This can be done by displaying all of the data in a structure and then using the tags to get the handles of the displayed objects you wish to remove. The first European example made use of this technique, as does the next one. It shows rivers, lakes, and coastlines. International boundaries have been removed using their tags:

```
clma
axesm mollweid; framem; gridm
displaym(P0line)
delete(handle('International Boundary'))
displaym(DN1line)
```

Here is the map of world coastlines and drainage:



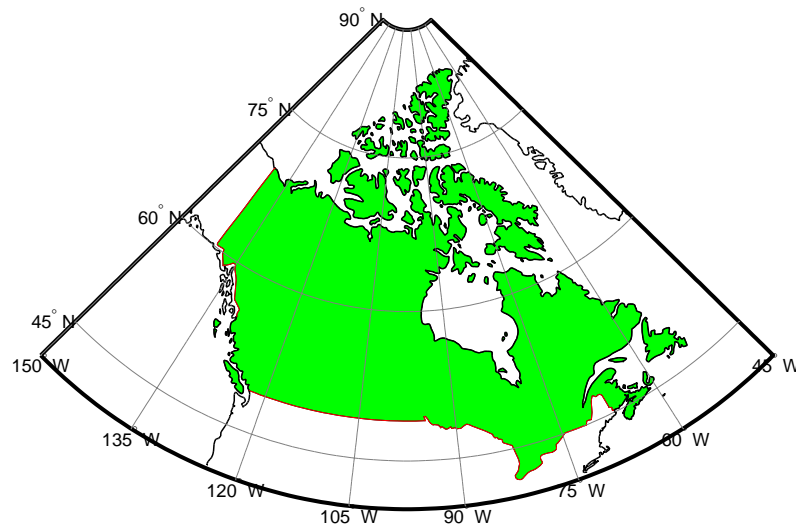
There are a number of other ways to work with the tags associated with this data. We can view the tags in the workspace by working directly with the geographic data structure.

```
unique(strvcat(DNLine.tag), 'rows')  
ans =  
Inland shorelines  
Streams, rivers, channelized rivers
```

Or we can use `mlayers` and `mobjects` to plot the structures, view the tags, and manipulate similarly tagged objects as a group. See Chapter 6, “GUI Tools” for some examples.

The tags can also be used to extract data from the structures. The following commands extract the patch data for Canada and plot it with the political line data:

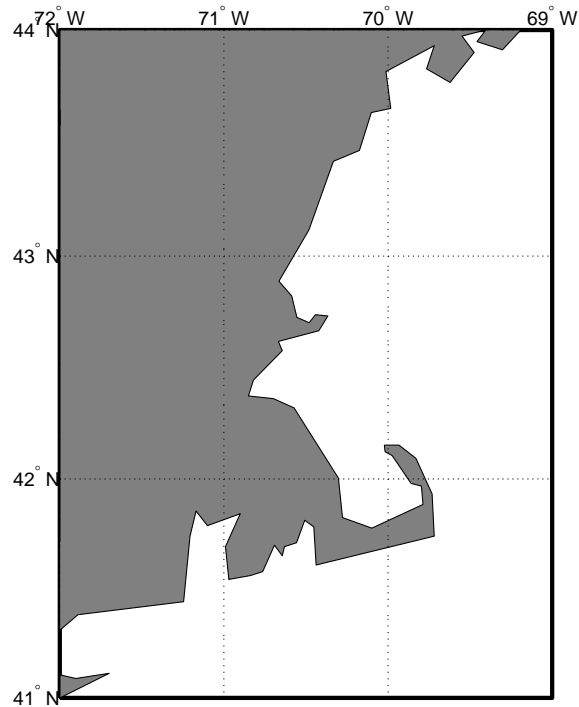
```
figure
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[40 90],'MapLonLimit',[-150 -45],...
      'MLabelLocation',15,'MLineLocation',15,...
      'PLabelLocation',15,'PLineLocation',15,...
      'GColor',.5*[1 1 1],'GLinestyle','-','...',...
      'MLabelParallel','south')
frame; gridm; mlabel; plabel
displaym(POLine)
[clat,clon] = extractm(P0patch,'canada');
patchesm(clat,clon,'g')
```



Extracting and displaying only the desired data can reduce the time and memory required to display a map. The Mapping Toolbox automatically trims and saves data outside the current map latitude and longitude limits. If you use `displaym` to show a small part of the world, the rest of the world data is retained within the map axes structure. Every time the projection parameters are changed, all of the world data is restored, trimmed, and projected. This overhead can be significant for a dataset as large as `worldo`.

Apply the extraction technique to generate a display of Cape Cod. As was shown in the earlier examples, this data contains somewhat generalized shapes for the countries and rivers. The level of detail is great enough for global and regional maps, but the distinct data points become noticeable when displayed for smaller regions:

```
figure
axesm('MapProjection', 'mercator', ...
      'MapLatLimit', [41 44], 'MapLonLimit', [-72 -69])
framem
gridm('MLineLocation', 1, 'PLineLocation', 1)
mlabel('MLabelLocation', 1)
plabel('PLabelLocation', 1)
[lat, lon] = extractm(P0patch, 'united states');
patchsm(lat, lon, 'FaceColor', .5*[1 1 1])
```



The `worldio` atlas data can also be used to look up names and locations. All of the names mentioned in the `P0text` and `PPtext` structures are collected in a `gazette` structure, which can be queried using `extractm`. For example, the location of the city of Antananarivo on the island of Madagascar can be found by using the following commands:

```
[lat, long, indx] = extractm(gazette, 'antan');
gazette(indx)
ans =
    type: 'line'
otherproperty: []
    tag: 'Antananarivo'
    string: 'Populated Place Name'
altitude: []
    lat: -18.6900
    long: 47.4480
```

A much more extensive gazette feature can be found in the Digital Chart of the World (DCW), containing more than 10,000 names, compared to about 500 in the `worldio` MAT-file. See Chapter 7, “External Data Interface” for more information on the DCW.

World Matrix Data

Political

The Mapping Toolbox includes a set of matrix data coded with political information in the MAT-file `worldmtx`.

```
load worldmtx
whos
```

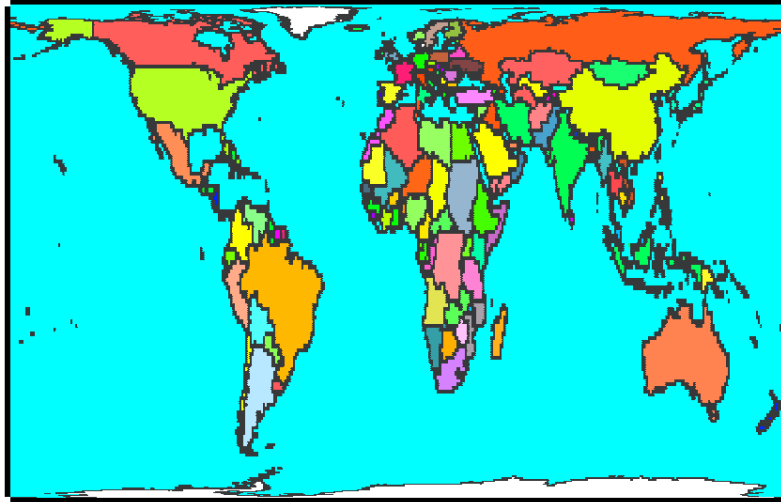
Name	Size	Bytes	Class
<code>clrmap</code>	195x3	4680	double array
<code>map</code>	180x360	518400	double array
<code>maplegend</code>	1x3	24	double array
<code>nations</code>	1x195	21324	struct array

The variable `map` is a regular matrix map at the same resolution (180 by 360) as the `topo` data, but in this case, it is an indexed map of political entities, as opposed to a matrix of elevations and bathymetric data. The `nations` structure relates the indices in the map to the names of countries, and the variable `clrmap` is a colormap that provides a good political display of the world.

Display the political regular matrix map using the `meshm` function and the provided colormap:

```
axesm gortho
meshm(map, maplegend); colormap(clrmap)
```


The map is shown in a Gall Orthographic projection, more recently made familiar as the Peters projection. This projection is equal-area, a property that is often desirable in representing competing political units.



The low resolution of this data makes it suitable only for global displays. Larger scale, regional maps are better made with vector data such as that in the worldio database. More detailed political matrix maps can be created from vector data using the techniques described in Chapter 6 of this document, “GUI Tools.”

This dataset can also be used to determine which country contains a point known by its latitude and longitude. What country claims sovereignty over the point at 5° E and 13° N?

```
code = ltl2val(map, maplegend, 5, 13)
code =
    34

nations(code)
ans =
    name: 'Cameroon'
```

Work through this example to see how the result was obtained. The matrix `map` contains integer country codes. By calling the function `l t l n2val`, we extracted the country code of the geographic point. The country code is an index into the `nations` structure, which returns the name of the country.

The coarseness of this dataset imposes some limitations. Just like any other matrix entry, the coastline and borders between nations are one degree quadrangles that may cover more than 100 kilometers on a side. The code for a border is often returned when you query areas with small nations. The actual shape of small countries is also poorly represented by such large cells. For instance, the area represented by the map in the `cape` MAT-file would cover just 9 cells for this dataset.

Terrain

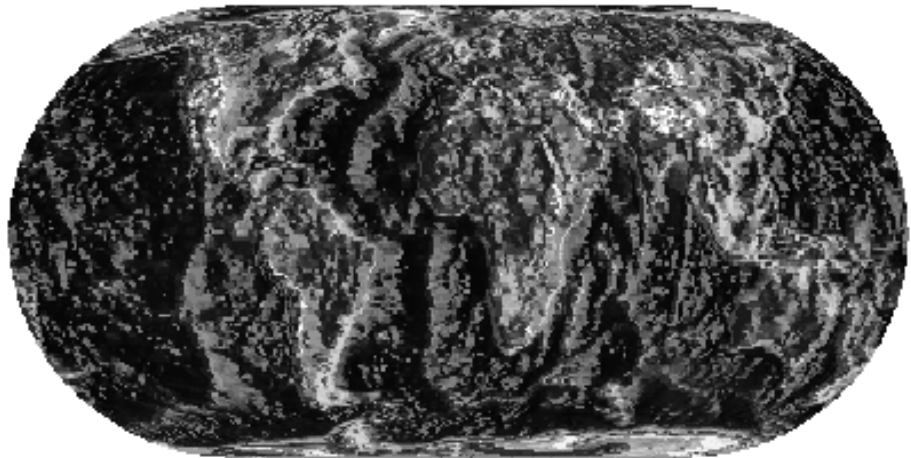
A low resolution set of global elevation data is provided with MATLAB in the topo workspace. It is a regular matrix of average elevations and depths (in meters) for a constant grid spacing of one degree by one degree.

```
load topo
whos
```

Name	Size	Bytes	Class
topo	180x360	518400	double array
topo legend	1x3	24	double array
topomap1	64x3	1536	double array
topomap2	128x3	3072	double array

Here it is displayed in an equal-area Eckert IV projection as a shaded-relief surface, which allows both large and small features to be seen:

```
axesm eckert4
meshl srm(topo, topo legend)
```



United States Vector Data

Low Resolution Data

Vector political and coastline data for the United States are provided in a number of useful formats in the `usalo` workspace.

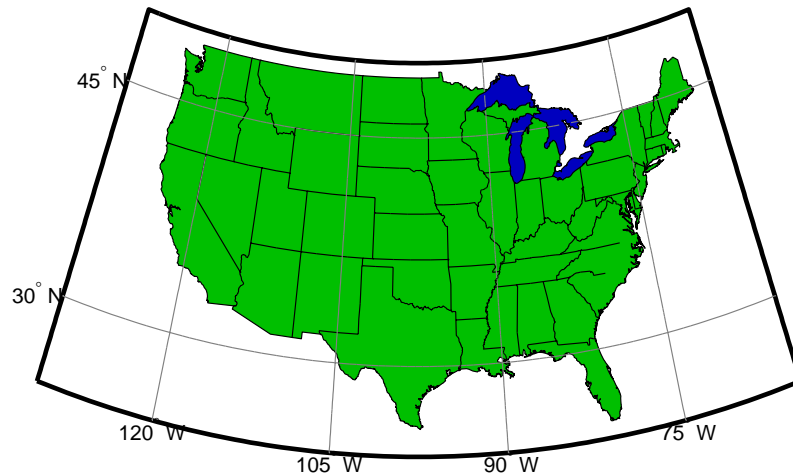
```
load usalo
whos
```

Name	Size	Bytes	Class
<code>conus</code>	1x1	71072	struct array
<code>greatlakes</code>	1x3	22220	struct array
<code>gtlakelat</code>	1229x1	9832	double array
<code>gtlakelon</code>	1229x1	9832	double array
<code>state</code>	1x51	256986	struct array
<code>stateborder</code>	1x1	38390	struct array
<code>statelat</code>	2345x1	18760	double array
<code>statelon</code>	2345x1	18760	double array
<code>uslat</code>	4339x1	34712	double array
<code>uslon</code>	4339x1	34712	double array

There are several types of data here. The double arrays are vectors of latitude and longitude points. The pair `uslat` and `uslon` form the outline of the coast and political borders of the continental United States. The polygon closes on itself, so it may be filled as a patch. Next, the `statelat` and `statelon` form the borders between the states for display as lines. Finally, `gtlakelat` and `gtlakelon` are patchable outlines of the Great Lakes. The commands suitable for displaying vector data of this type are `plotm`, `linem`, `fillm`, `patchm`, and `patchesm`.

Display the data vectors as patches and lines:

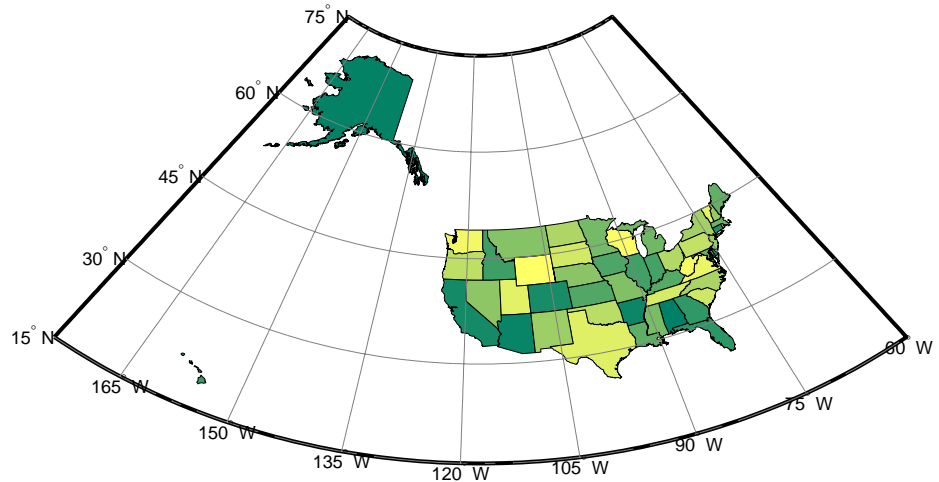
```
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[24 50],'MapLonLimit',[-130 -65],...
      'MLabelLocation',15,'MLineLocation',15,...
      'PLabelLocation',15,'PLineLocation',15,...
      'GColor',.5*[1 1 1],'GLineStyle','- ',...
      'MLabelparallel','south')
framem; gridm; mlabel; plabel
patchm(uslat,uslon,[0 .75 0])
patchm(gtlat,gtlon,1,[0 0 .75])
plotm(statelat,statelon,'k')
```



The structure arrays are Mapping Toolbox geographic data structures and can be displayed using `mlayers` or `displm`. The data in `stateborder` is identical to that in `statelat` and `statelon`. Similarly, `conus` contains the same outline of the continental United States as `uslat` and `uslon` but is defined to be displayed as a patch. The 50 state and District of Columbia patches are located in the `state` structure.

Display the state structure using `displaym`. Notice the only one command is needed to plot 51 different patches.

```
figure
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[15 75],'MapLonLimit',[-175 -60],...
      'MLabelLocation',15,'MLineLocation',15,...
      'PLabelLocation',15,'PLineLocation',15,...
      'GColor',.5*[1 1 1],'GLineStyle','- ',...
      'MLabelParallel','south')
frame; gridm; mlabel; plabel
displaym(state); colormap(summer)
```



The states are all tagged, allowing them to be manipulated or to be extracted by name. You can see the available state names by entering the following:

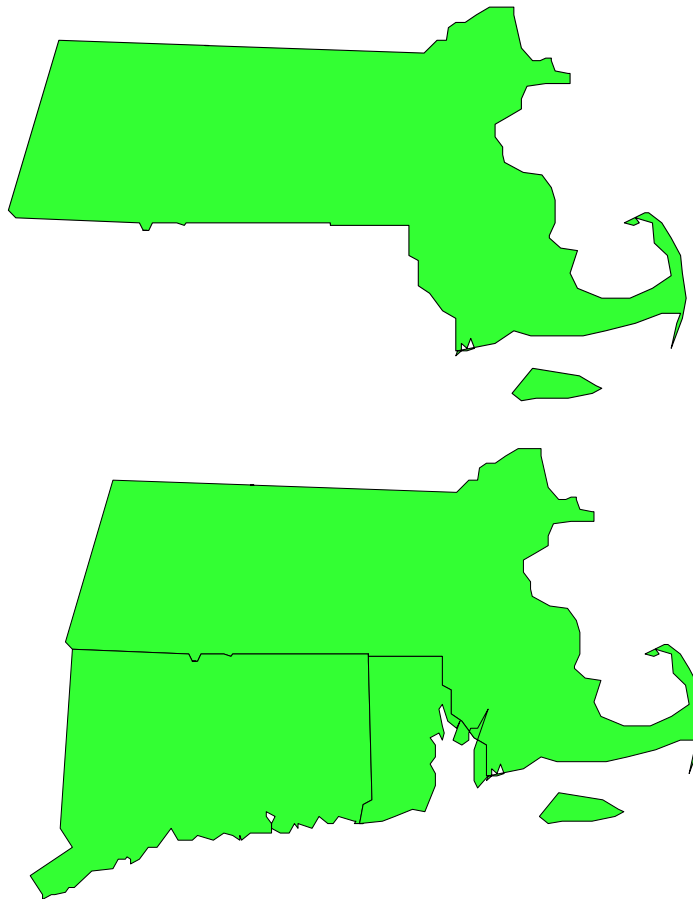
```
strvcat(state.tag)
```

The state of Massachusetts can be displayed by itself or with its neighboring southern states, Connecticut and Rhode Island:

```
[lat1, long1] = extractm(state, 'mass');  
[lat2, long2] = extractm(state, strvcat('mass', 'conn', 'rhode'));
```

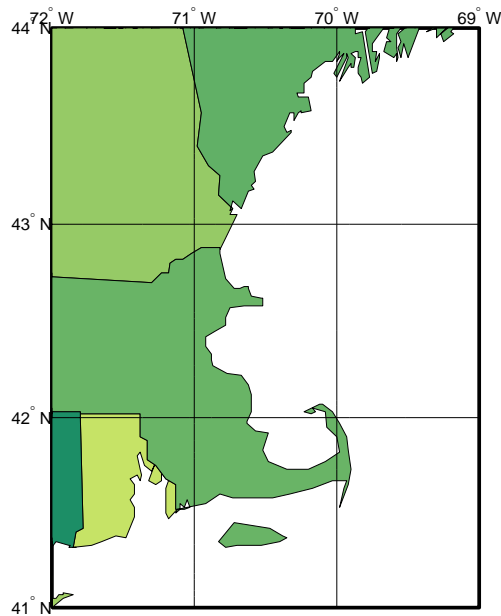
```
figure; axesm mercator  
patchm(lat1, long1, [0 .75 0])
```

```
figure; axesm mercator  
patchsm(lat2, long2, [0 .75 0])
```



Finally, look at the region around Cape Cod again, to get an impression of the detail of the data:

```
figure
axesm('MapProjection', 'mercator', 'Frame', 'on', ...
      'MapLatLimit', [41 44], 'MapLonLimit', [-72 -69])
gridm('MLineLocation', 1, 'PLineLocation', 1, ...
      'GLineStyle', '-')
mlabel('MLabelLocation', 1)
plabel('PLabelLocation', 1)
displaym(state); colormap(summer)
```



There is an obvious compromise between the size of a dataset and its accuracy and detail. This data can be displayed relatively quickly for displays of all the United States, but it is not suitable for mapping smaller regions. Note the generalized character of the coastline and the absence of small islands. More detailed and accurate data is available in the `usahi` workspace and through the Mapping Toolbox External Data Interface functions.

Medium Resolution State Outlines

Another more detailed set of state outlines is available in the Mapping Toolbox. The data in the `usahi` MAT-file is similar in format to the `state` structure found in `usalo` but contains more detailed coastlines and islands. The resolution of `stateline` in the `usahi` workspace is about three times greater than the corresponding `state` in the `usalo` workspace.

```
load usahi
```

```
whos
```

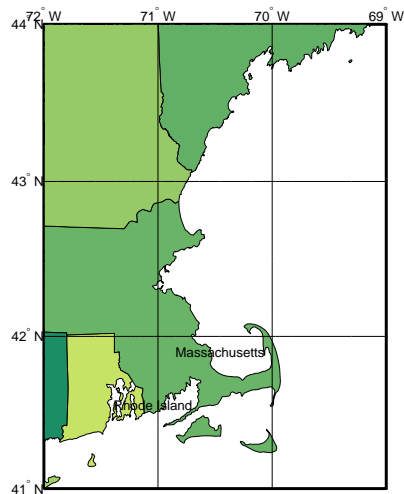
Name	Size	Bytes	Class
<code>statelin</code>	1x51	837656	struct array
<code>statepatch</code>	1x51	837758	struct array
<code>statetext</code>	1x51	51190	struct array

The listed variables are all formatted geographic data structures containing lines, patches, or text. The data in the `statelin` structure is the same as that in `statepatch` but has been defined to display as lines rather than patches. The structure `statetext` contains the corresponding names of the states. Note that because of the higher resolution of this data, it may require more than the default memory size to display and will take longer to project and render. This difference is particularly marked for patches.

Display the map in an Equal-Area Conic projection:

```
axesm('MapProjection','eqaconic','mapparallels',[],...
      'MapLatLimit',[15 75],'MapLonLimit',[-188 -60],...
      'MLabelLocation',15,'MLineLocation',15,...
      'PLabelLocation',15,'PLineLocation',15,...
      'GColor',.5*[1 1 1],'GLineStyle','- ',...
      'MLabelParallel','south')
frame; gridm; mlabel; plabel
display(statepatch); colormap(summer)
```

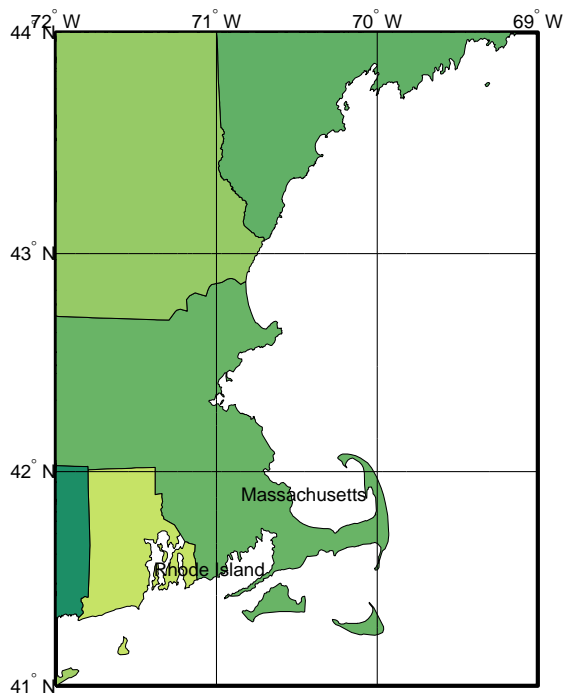
Here is the result:



At this scale, it is difficult to see the difference between this data and the much smaller usual workspace. By focusing on the Cape Cod region, the higher resolution becomes more apparent:

```
figure
axesm('MapProjection', 'mercator', 'Frame', 'on', ...
      'MapLatLimit', [41 44], 'MapLonLimit', [-72 -69])
gridm('MLineLocation', 1, 'PLineLocation', 1, 'GLineStyle', '- -')
mlabel('MLabelLocation', 1)
plabel('PLabelLocation', 1)
displaym(statepatch)
displaym(statetext)
colormap(summer)
```

Here is the zoomed-in region of Cape Cod. The level of detail is noticeably greater than that for the data in the usual MAT-file.



United States Matrix Data

Political

The Mapping Toolbox also provides a set of political matrix data for the continental United States located in the `usamtx` MAT-File.

```
load usamtx
whos
```

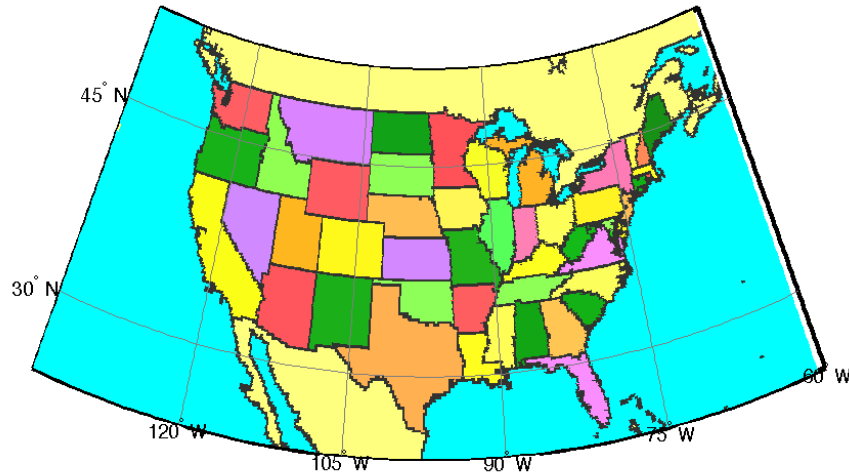
Name	Size	Bytes	Class
<code>clrmap</code>	54x3	1296	double array
<code>map</code>	140x370	414400	double array
<code>maplegend</code>	1x3	24	double array
<code>states</code>	54x15	1620	char array

The data is in a regular matrix map format, similar to that discussed in the section “World Matrix Data.” The variable `map` is at a resolution of 5 cells per degree, or roughly 20 kilometers or better. Recall that the resolution of the `worldmtx` data is 1 cell per degree; hence, this dataset is five times finer. A `colormap` has been provided, along with a vector of state names, both corresponding to the political code, or “index,” used in the map matrix.

Display the matrix map in an Albers Equal-Area Conic projection:

```
axesm('MapProjection','eqaconic','MapParallels',[],...
      'MapLatLimit',[24 52],'MapLonLimit',[-134 -60],...
      'MLabelLocation',15,'MLineLocation',15,...
      'PLabelLocation',15,'PLineLocation',15,...
      'GColor',.5*[1 1 1],'GLinestyle','- ',...
      'MLabelParallel','south')
framem; gridm; mlabel; plabel
meshm(map,maplegend); colormap(clrmap)
```

Here is the result:



As with the world matrix data, the low resolution of this data makes it suitable for only large area displays. Vector maps are more appropriate where more detail is required. Chapter 6, “GUI Tools,” describes how to create higher resolution matrix maps from vector maps.

This dataset can also be used to determine which state contains a particular geographic point. In what state is Alamogordo (32.8990° N and 105.9570° W)?

```
code = tl2val(map, maplegend, 32.8990, -105.9570)
code =
    34

states(code, :)
ans =
New Mexico
```

Terrain

As an example of a higher resolution digital elevation map, MATLAB provides the `cape` workspace, containing an image of elevation data for the northeastern United States on a 30 arc-second grid (resolution of about one kilometer or better on the ground). The data can be defined as a regular matrix map using the `loadcape` function script, which rearranges the data and provides the necessary map legend:

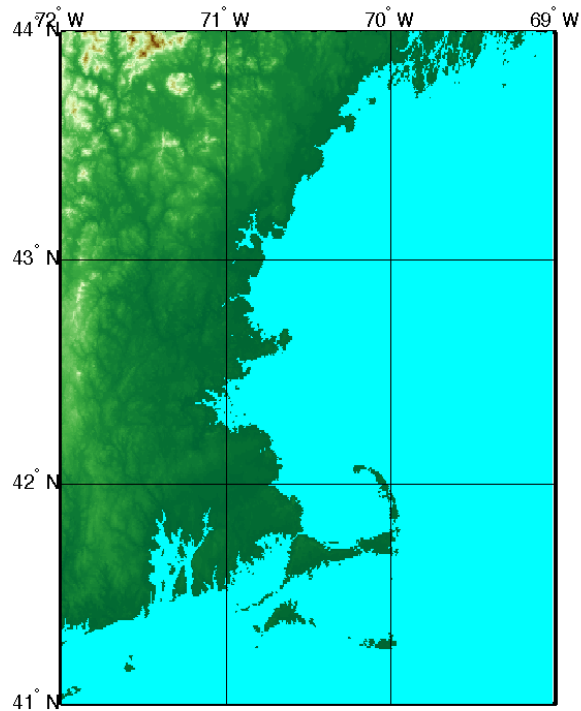
```
loadcape
whos
```

Name	Size	Bytes	Class
caption	2x55	220	char array
cmap	192x3	4608	double array
map	360x360	1036800	double array
maplegend	1x3	24	double array

Here we display the elevation data using a conformal Mercator projection, so shapes of small regions suffer little distortion, while the distortion in relative areas is scarcely noticeable for such a small region.

```
axesm('MapProjection','mercator',...
      'MapLatLimit',[41 44], 'MapLonLimit', [-72 -69])
framem
gridm('MLineLocation',1, 'PlineLocation',1, 'GLineStyle',' - ')
mlabel('MLabelLocation',1)
plabel('PLabelLocation',1)
meshm(map, maplegend); demcmap(map)
```

Here is the result:



Global coverage of digital terrain and bathymetry at this resolution is provided through the Mapping Toolbox External Data Interface. A variety of freely available digital elevation maps is available over the Internet for import into MATLAB. These maps range in resolution from about 10 km to 100 meters. See Chapter 7, “External Data Interface” for more information on the data and import functions.

Astronomical Data

Although the Earth may be the most commonly mapped object, the same cartographic techniques are used to map the stars and planets. As an example of such astronomical data, the Mapping Toolbox includes a map of the stars in the stars workspace:

```
load stars
whos
```

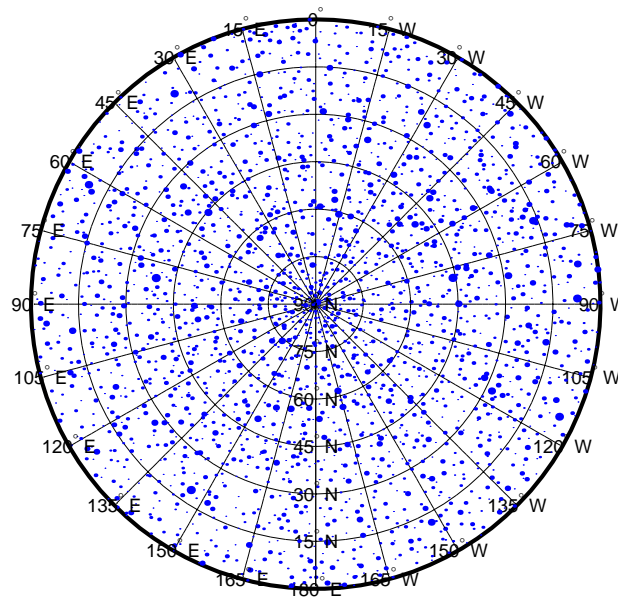
Name	Size	Bytes	Class
gl at	4652x1	37216	double array
gl ong	4652x1	37216	double array
l at	4652x1	37216	double array
l ong	4652x1	37216	double array
pvmag	4652x1	37216	double array
vmag	4652x1	37216	double array

This is a set of more than 4500 star locations and visual magnitudes derived from the Fifth Fundamental Catalog of Stars parts I and II (FK5 and FK5e). Other star data such as mean errors, proper motions, spectral types, parallaxes, radial velocities, and cross identifications to other catalogues can be obtained from the catalog using the interface function described in the “External Data Interface” chapter.

The positions of stars in equatorial coordinates are given in the vectors `l at` and `l ong`. These are latitudes and longitudes in degrees for the stars as seen from within the celestial sphere. The visual magnitude of each star is given in the vector `vmag`. Stars are typically plotted with diameters proportional to the visual magnitude. The vector `pvmag` contains modified `vmag` data suitable for display with `scatterm` to exhibit this proportional diameter. All magnitudes are positive, and brighter stars have a larger `pvmag` number.

Display the stars of the northern sky in the Equidistant azimuthal projection:

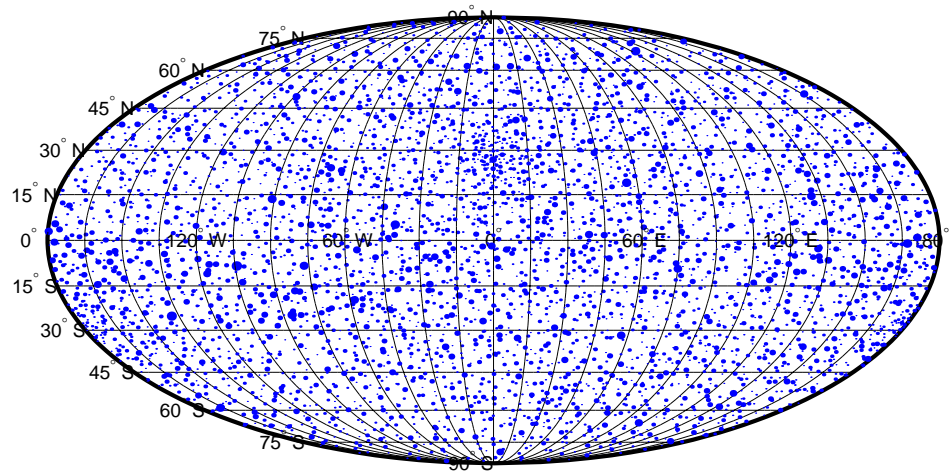
```
axesm eqdazi m
framem; gridm; mlabel; plabel
setm(gca, 'Origin', [90 180 0], 'FlatLimit', [-inf 90], ...
      'MLabelLocation', -180:15:165, 'MLineLocation', 15, ...
      'MLineLimit', [-75 75], 'MLabelParallel', 'equator', ...
      'PLabelLocation', 15:15:75, 'GLineWidth', .01, ...
      'GLineStyle', '- -')
scatterm(lat, long, pvmag, ' . ', 'MarkerSize', .0125)
```



You may be able to identify the Big Dipper, or Ursa Major constellation, located at 180° E, 60° N.

Another system used to describe the positions of heavenly bodies is galactic coordinates. This coordinate system puts the center of our galaxy at the origin and places the north pole so that the Milky Way is aligned with the galactic equator. The positions of the stars in this coordinate system are given in the vectors `glat` and `glong`. Stars in galactic coordinates are typically plotted in a pseudo-cylindrical projection like the Mollweide:

```
figure
axesm mollweid
frame; gridm; mlabel; plabel
setm(gca, 'FlatLimit', [-inf 90], 'GLineWdth', .01, ...
      'GLineStyle', '-', 'MLabelParallel', 'equator', ...
      'MLabelLocation', -120:60:180, 'MLineLocation', 15)
scatterm(glat, glong, pvmag, 'b', 'MarkerSize', .0125)
```



See Chapter 7, “External Data Interface” for more information on how to convert between these two systems.

Map Projections

What Is a Projection?	4-2
Quantitative Properties	4-3
Geometric Surfaces	4-4
Projection Aspect	4-7
The Origin Vector	4-7
Coordinate Transformations	4-14
Projection Computations	4-18
Summary Guide	4-22

What Is a Projection?

It has long been established that the shape of the Earth resembles a sphere and not a flat surface. If the world were indeed flat, cartography would be a trivial matter and map projections would be quite unnecessary.

To create a two-dimensional representation of a curved surface such as the Earth, you must determine how to transform that surface to a plane. The method of this transformation is called a *map projection*, from the geometric methods that were traditionally used to construct maps. While many map projections no longer rely on physical projections, it is useful to think of map projections in geometric terms.

Quantitative Properties

A sphere, unlike a cone or cylinder, cannot be simply reformed into a plane. In order to view the surface of a round body on a two-dimensional flat plane, you must first define a developable surface (i.e., one that can be “cut” and “flattened” onto a plane without stretching) and devise a means of systematically representing all or part of the spherical surface onto the plane. This inevitably leads to distortions of one kind or another. Three characteristic properties of map projections are subject to distortion: area, shape, and scale. A given projection cannot retain more than one of these properties over a large area of the Earth.

An *equal-area* projection is one in which equal spatial units at different points on the map represent the same true area measurement. For example, consider a coin of any size as our spatial unit. The area of the Earth covered by the coin on one part of the map display is exactly equal to that covered by the coin on any other part. Shape and scale must be distorted, although there are some equal-area maps designed to minimize these distortions for at least parts of the map and perhaps preserve shape or scale at specific points.

Other terms for equal-area projections include *equivalent*, *homolographic* or *homalographic* (from the Greek *homalos* or *homos*, “same,” and *graphos*, “write”), *authalic* (from the Greek *autos*, “same,” and *ailos*, “area”), and *equireal*.

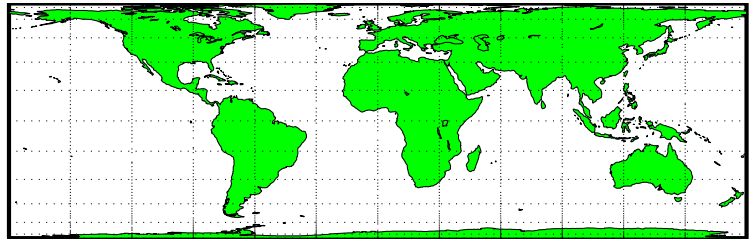
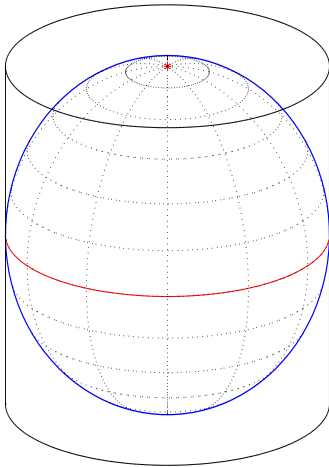
A *conformal* projection is one that preserves the relative local angles (shape) about every point on the map, except for one or more possible “singular” points. Since relative local angles are correct, meridians intersect parallels at right angles (90 degrees). No map can be both equal-area and conformal. Another term for conformal is *orthomorphic* (from the Greek *orthos*, “straight,” and *morphe*, “shape”).

An *equidistant* projection shows true scale between one or two points and every other point on the map, or along every meridian. No map projection shows scale correctly in all directions throughout the entire map.

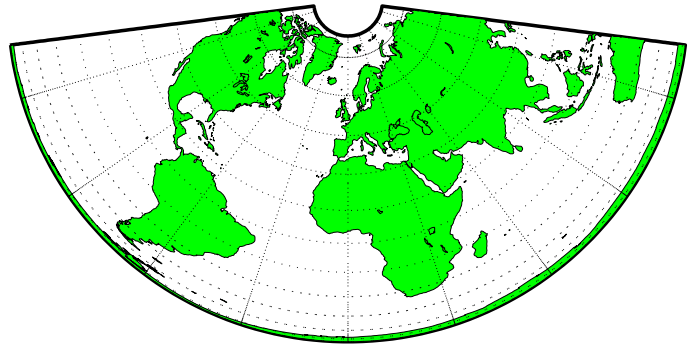
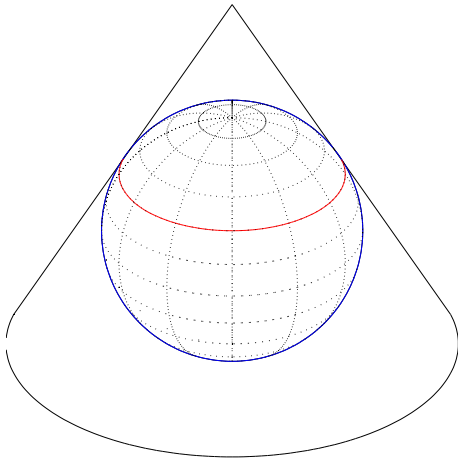
Geometric Surfaces

Three standard types of geometric surfaces are used to develop map projections: the cylinder, the cone, and the plane. A few projections, however, cannot be categorized as such, or are combinations of these. The three classifications are used for a wide variety of projections, including some that are not geometrically constructed.

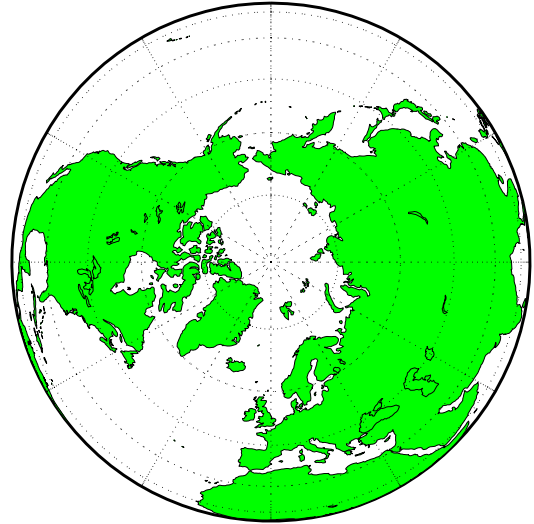
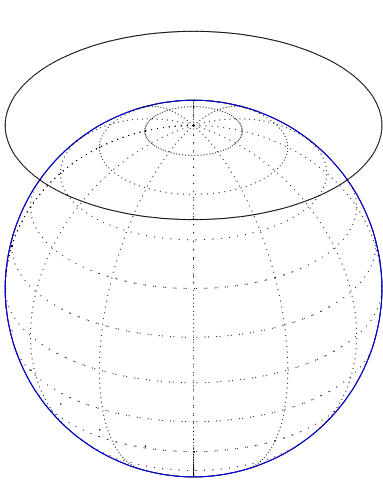
A *cylindrical* projection is produced by wrapping a cylinder around a globe representing the Earth. The map projection is the image of the globe projected onto the cylindrical surface, which is then unwrapped into a flat surface. Parallels appear as horizontal lines and meridians as vertical lines. The following shows a regular cylindrical or “normal aspect” orientation in which the cylinder is tangent to the Earth along the Equator and the projection is calculated horizontally from the axis.



A *conic* projection is derived from the projection of the globe onto a cone placed over it. For the “normal aspect,” the apex of the cone lies on the polar axis of the Earth, and the surface of the cone touches the globe along a particular parallel of latitude. A *polyconic* projection considers each band of parallels as part of separate cones, each tangent to the globe at that particular parallel latitude.



An *azimuthal* projection is a projection of the globe onto a plane. For a polar azimuthal projection, the plane is tangent to the Earth at one of the poles, with meridians projected as straight lines radiating from the pole, and parallels shown as complete circles centered at the pole.



Each of these projection types can be further modified:

- Instead of being tangent to one standard parallel, the cylindrical or conic surface may cut the globe at two parallels. In like manner, the plane for an azimuthal projection may intersect the globe instead of being tangent.
- The vantage point of the projection onto the geometric surface may be changed. The perspective may be chosen from the center of the Earth or along the polar axes (*central* or *gnomonic*), from the opposite side of the Earth's surface (*stereographic*), or from an infinite point in space (*orthographic*).

Projection Aspect

Until now, the discussion of map displays has focused on the *normal* aspect, by far the most commonly used. This section discusses the use of *transverse*, *oblique* and, *skew-oblique* aspects.

Projection aspect is primarily of interest in the display of maps. However, this section will also discuss how the idea of projection aspect as a coordinate system transformation can be applied to map variables for analytical purposes.

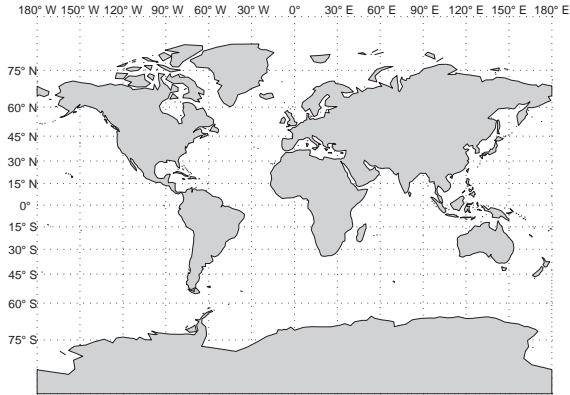
The Origin Vector

A map axes `Origin` property is a vector describing the geometry of the displayed projection. The form of this vector is of the following:

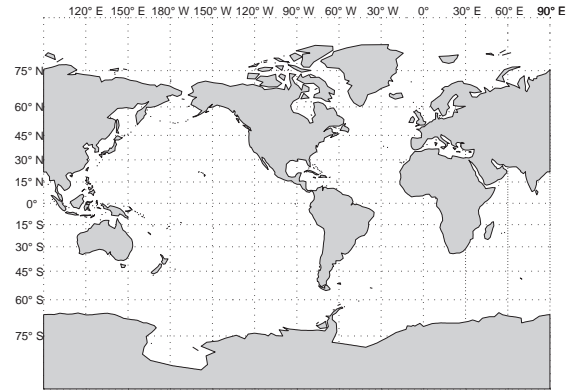
$$\text{origvec} = [\textit{latitude} \ \textit{longitude} \ \textit{orientation}]$$

The latitude and longitude represent the geographic coordinates of the center point of the display from which the projection is calculated. The orientation refers to the angle from “straight up” at which the North Pole bears from this center point. The default origin vector is [0 0 0], that is, the projection is centered on the geographic point (0°,0°) and the North Pole is “straight up” from this point. Such a display is in a *normal* aspect. In fact, changes to only the longitude value of the origin vector do not change the aspect; a *normal* aspect is one centered on the Equator in latitude with an orientation of 0°.

Both of these Miller projections have normal aspects, despite having different origin vectors:



Origin at (0°, 0°), with a 0° Orientation.
(origin vector = [0 0 0])



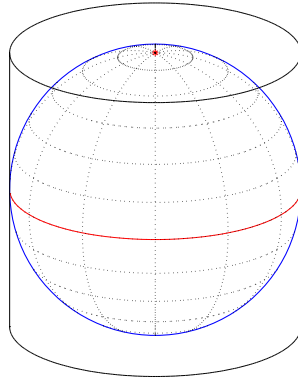
Origin at (0°, 90°W), with a 0° Orientation.
(origin vector = [0 -90 0])

This makes sense if you think about a simple, true cylindrical projection. This is the projection of the globe onto a cylinder wrapped around it. For normal aspects, this cylinder is tangent to the globe at the Equator, and changing the origin longitude simply corresponds to rotating the sphere about the longitudinal axis of the cylinder. If we continue with the wrapped-cylinder model, we can understand the other aspects as well.

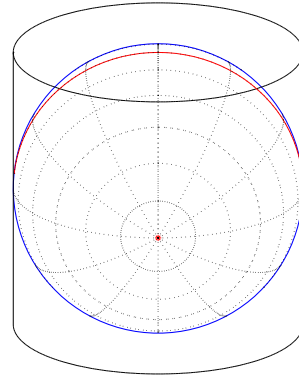
Following this description, a *transverse* projection can be thought of as a cylinder wrapped around the globe tangent at the poles and along a meridian. Finally, when such a cylinder is tangent along an arbitrary great circle, the result is an *oblique* projection.

Here are diagrams of the four cylindrical map orientations, or aspects:

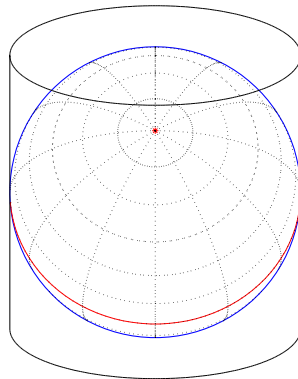
Normal



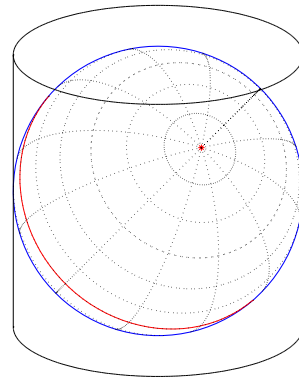
Transverse



Oblique



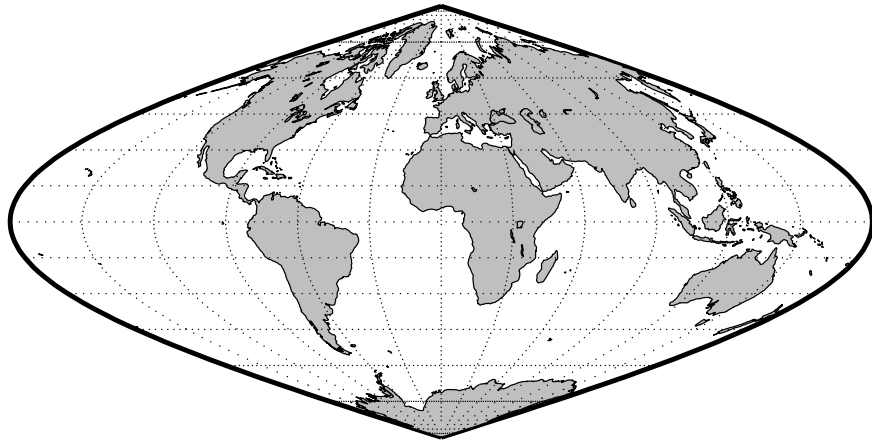
Skew-Oblique



Of course, few projections are actually true cylindrical projections, but the concept of the wrapped cylinder is nonetheless a convenient way to think about it. Perhaps the best way to gain an appreciation for projection aspect is to look at a few examples.

NOTE: The projection aspect discussed in this section is different from the map axes Aspect property. The map axes Aspect property controls the orientation of the figure axes. For instance, if a map is in a 'normal' setting with a "landscape" orientation, a switch to a 'transverse' aspect rotates the axes by 90°, resulting in a "portrait" orientation. The projection aspect deals with the display "orientation" of the map projection. One cannot change the projection aspect by simply altering the Aspect property; the Origin vector must be used instead.

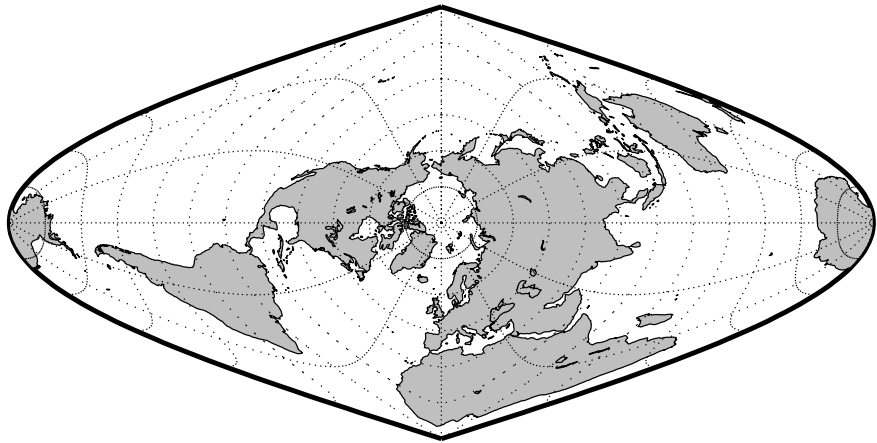
Here, a pseudocylindrical projection, the Sinusoidal, is used. First, look at the normal aspect of this projection:



Normal Aspect: Origin at $(0^\circ, 0^\circ)$, Orientation 0°
(origin vector = $[0\ 0\ 0]$)

In the normal aspect, the North Pole is at the "top" of the image. To create a transverse aspect, imagine "pulling" the North Pole down to the center of the image, which was originally occupied by the point $(0^\circ, 0^\circ)$.

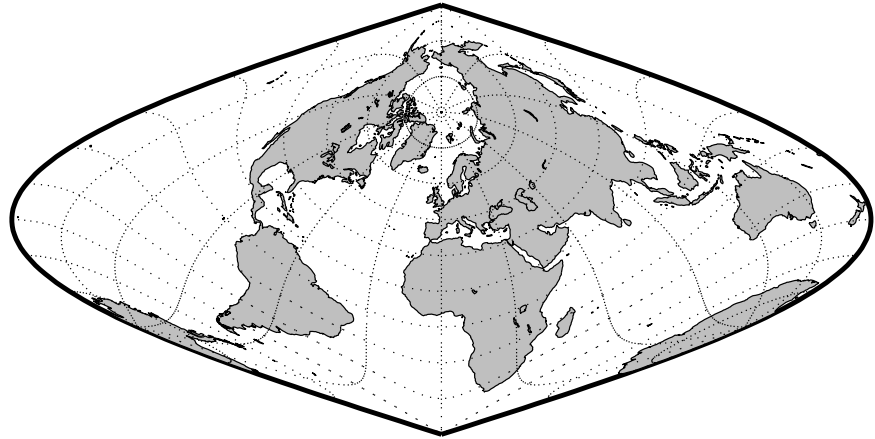
The shape of the frame is unaffected; this is still a Sinusoidal projection:



Transverse Aspect: Origin at $(90^{\circ}\text{N}, 0^{\circ})$, Orientation 0°
(origin vector = $[90\ 0\ 0]$)

The normal and transverse aspects can be thought of as limiting conditions. Anything else is an oblique aspect. Conceptually, if you push the North Pole halfway back to its original position, that is, to the position originally occupied by the point $(45^{\circ}\text{N}, 0^{\circ})$ in the normal aspect, the result will be a simple oblique aspect.

Here is the oblique aspect Sinusoidal projection:

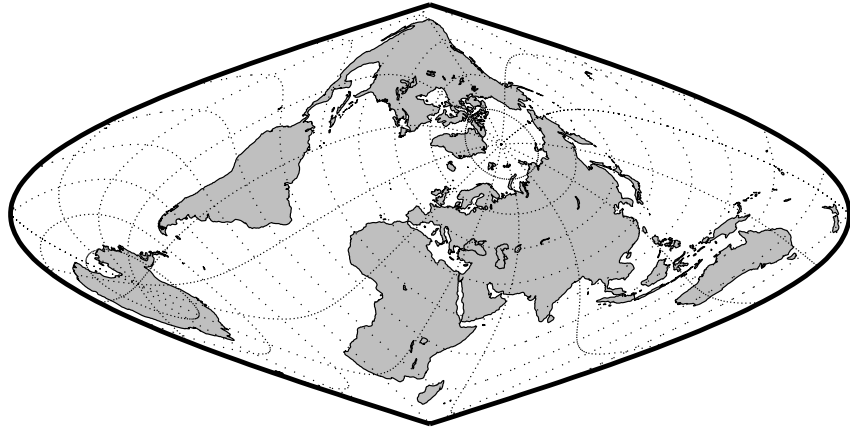


Oblique Aspect: Origin at $(45^{\circ}\text{N}, 0^{\circ})$, Orientation 0°
(origin vector = $[45\ 0\ 0]$)

Another way of understanding this is to imagine the new origin point $(45^{\circ}\text{N}, 0^{\circ})$ being pulled to the center of the image, to the point originally occupied by $(0^{\circ}, 0^{\circ})$ in the normal aspect.

The examples so far have left the aspect orientation at 0° . If the orientation is altered, an oblique aspect becomes a *skew-oblique*. Imagine the previous example with an orientation of 45° . Conceptually, the skew-oblique aspect corresponds to pulling the new origin point, $(45^{\circ}\text{N}, 0^{\circ})$ down to the center of the projection and then rotating the projection until the North Pole lies at an angle of 45° clockwise from straight up with respect to the new origin.

Here is the result:



Skew-Oblique Aspect: Origin at (45°N,0°), Orientation 45°
(origin vector = [45 0 45])

Any projection can be viewed in alternate aspects. Some of these are quite useful. For example, the transverse aspect of the Mercator projection is widely used in cartography, especially for mapping regions with predominantly north-south extent. One candidate for such handling might be Chile. Oblique Mercator projections might be used to map long regions that run neither north and south nor east and west, such as New Zealand.

The features of a projection are maintained in any aspect, *relative to the base projection*. As we have seen, the “outline,” or frame doesn’t change. Non-directional features also do not change. For example, the Sinusoidal projection is equal-area, and is so in any aspect. Directional features must be considered carefully, however. In the normal aspect of the Sinusoidal projection, scale is true along every parallel and the central meridian. This is not the case for the skew-oblique aspect; however, scale is true along the lines where these parallels and meridian would be in the corresponding base projection.

The base projection can be thought of as a standard coordinate system, and the normal aspect conforms to it. The other aspects can be thought of as coordinate transformations.

Coordinate Transformations

Previously, we discussed the concept of altering the display aspect of a map projection. One way to think of this is to consider redefining the coordinate system, and then displaying a projection based on the new system. For example, think about redefining your coordinate system so that your hometown is the origin. If you calculated a map projection in a normal aspect with respect to this “transformed” coordinate system, the resulting display would look like an oblique aspect with respect to the “true” coordinate system.

This transformation of coordinate systems can be useful independent of map displays. If a transformed coordinate system is selected so that your home town is the new “north pole,” then the transformed coordinates of all other points will provide interesting information.

The distance of each point from your hometown is then simply 90° minus the transformed latitude. For example, the point antipodal to your hometown would be the transformed “south pole.” -90° . Its distance from your hometown is then $90^\circ - (-90^\circ)$, or 180° , as expected. Points 90° distant from your hometown all have a transformed latitude of 0° . In fact, these points make up the transformed “equator.” The transformed longitudes correspond to the great circle azimuths of those points from your hometown.

When regular matrix maps are transformed in this manner, distance and azimuth calculations with the map variable reduce to row and column operations.

Vector Data – rotatem

As an example of transforming a coordinate system, suppose you live in Midland, Texas, at (32°N,102°W). You have a brother in Tulsa (36.2°N,96°W) and a sister in New Orleans (30°N,90°W).

```
mi dl _l at = 32;    mi dl _l on = -102;
tul s _l at = 36. 2; tul s _l on = -96;
newo _l at = 30;    newo _l on = -90;
```

What can we say about their homes, relative to yours?

```
di st2tul s = di stance(mi dl _l at, mi dl _l on, tul s _l at, tul s _l on)
di st2tul s =
        6. 5032
```

```
di st2newo = di stance(mi dl _l at, mi dl _l on, newo _l at, newo _l on)
di st2newo =
        10. 4727
```

Tulsa is about 6.5 degrees distant, New Orleans about 10.5 degrees distant. The great circle azimuths are the following:

```
azi2tul s = azi muth(mi dl _l at, mi dl _l on, tul s _l at, tul s _l on)
azi2tul s =
        48. 1386
```

```
azi2neworl = azi muth(mi dl _l at, mi dl _l on, newo _l at, newo _l on)
azi2neworl =
        97. 8644
```

The absolute difference in these azimuths is 49.7258°, a fact we will use later.

Today, you feel on top of the world, so make Midland, Texas, the “north pole” of a transformed coordinate system. To do this, first determine the origin required to put Midland at the pole:

```
ori gi n = newpol e(mi dl _l at, mi dl _l on)
ori gi n =
        58      78      0
```

The origin of the new coordinate system will be (58°N,78°E). Midland is now at a “new latitude” of 90°.

What are the “new coordinates” of Tulsa and New Orleans? To calculate these, use the `rotatem` command. Since it defaults to radians, be sure to include the units input 'degrees' :

```
[tul_s_lat1, tul_s_lon1] = rotatem(tul_s_lat, tul_s_lon, ...
                                origin, 'forward', 'degrees')

tul_s_lat1 =
    83.4968
tul_s_lon1 =
   -48.1386

[newo_lat1, newo_lon1] = rotatem(newo_lat, newo_lon, ...
                                origin, 'forward', 'degrees')

newo_lat1 =
    79.5273
newo_lon1 =
   -97.8644
```

Now, test the new coordinate system for the advertised properties. The distance to Tulsa was 6.5032° distant. Its colatitude in the new coordinate system is 90°-83.4968°, or 6.5032°. Similarly, the new colatitude of New Orleans, 90°-79.5273°, is the distance 10.4727°.

Additionally, the absolute difference in the azimuths of the two cities from Midland was 49.7258°. The difference in their “new longitudes” is -48.1386°-(-97.8644°), which is also 49.7258°, as advertised.

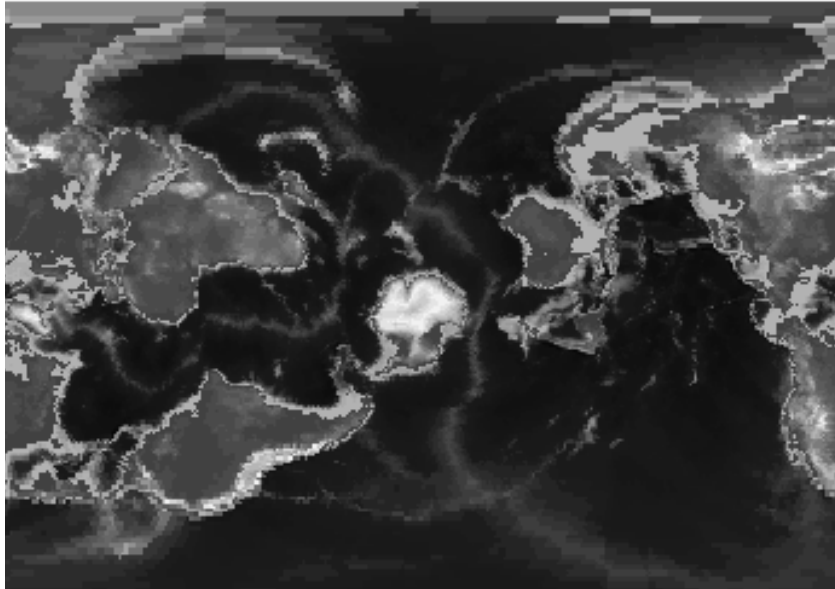
Matrix Data – neworig

A regular matrix map can be used to create a new general matrix map with its data rearranged to correspond to a new coordinate system. Suppose you want to transform the topo data to a new coordinate system in which Sri Lanka (7°N,80°E) is the “North Pole.”

```
load topo
origin = newpole(7, 80);
[map, lat, lon] = neworig(topo, toplotend, origin);
```

Display the new map:

```
axesm miller  
surfm(map, [30 30]); demcmap(topo)
```



An interesting feature of this new matrix map is that every cell in its first row is 0° - 1° distant from the Sri Lankan point (7°N , 80°E), and every cell in its second row is 1° - 2° distant, etc. Another feature is that every cell in a particular column has the same great circle azimuth from the point.

Projection Computations

Most of the examples in this document assume that the end product of a map projection is a graphical representation of the map. While this is true for most cases, there may be times when you need access to the non-geographic coordinates of your map data in the projected space.

An easy way to retrieve the projected coordinates of a map that has already been displayed is with the MATLAB `get` command. The projected coordinates are stored in the object's `xdata` and `ydata` properties. As an example, display a map frame for the Mollweide projection, and extract its x-y coordinates:

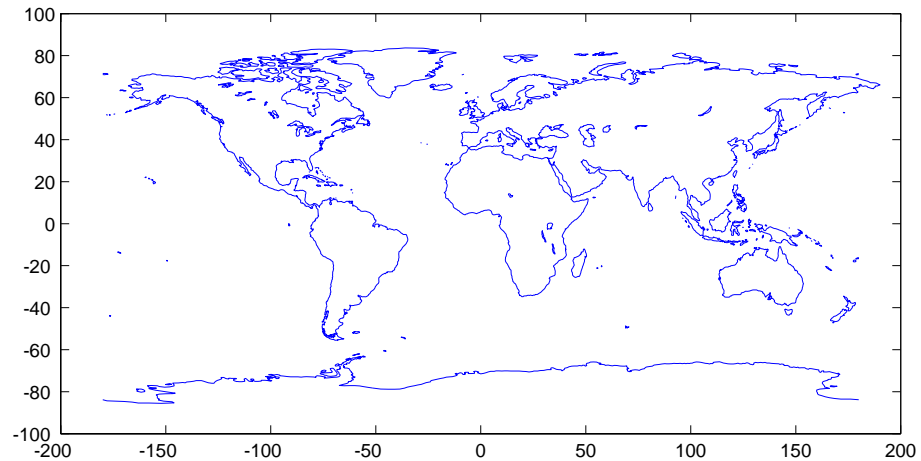
```
axesm mollweid
h = framem;
x = get(h, 'XData');
y = get(h, 'YData');
```

Of course, you do not need to display a map object to get its projected coordinates. You can perform the same projection computations that are done within the Mapping Toolbox display commands.

Begin by displaying a map of the coast vector data:

```
figure
load coast
plot(long, lat); axis equal
set(gca, 'XLim', [-200 200], 'YLim', [-100 100])
```

Here is the result. Note that some of the data extends beyond 180° longitude.



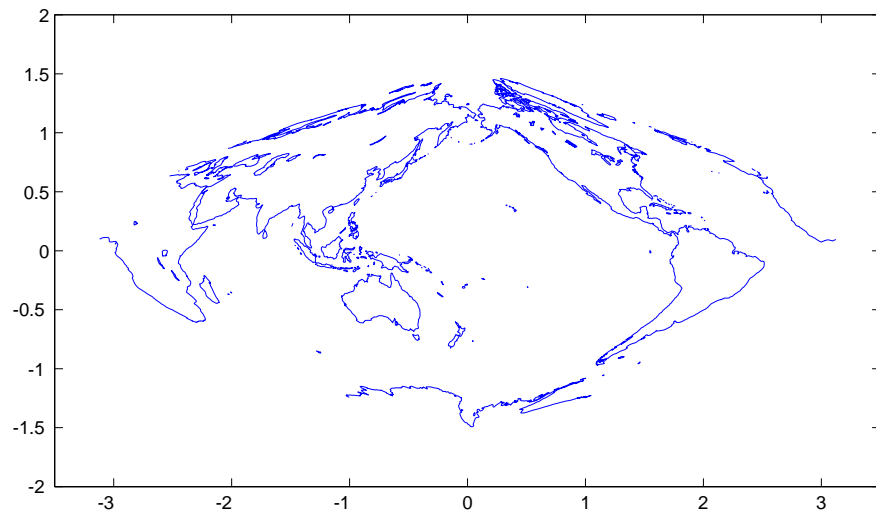
Before projecting the data, you must define the projection parameters, just as you would prepare a map axes with `axesm` before displaying a map. The projection parameters are stored in a map projection structure that normally resides in the `UserData` property of a MATLAB axes, but you can use it directly for the projection computation.

The following commands create an empty map projection structure for a Sinusoidal projection, change the map origin, and fill in the rest of the structure fields with default property values. Just as you can change the property settings of a map axes with `setm`, you can use the entries of the map projection structure to control the projection properties.

```
mstruct = defaultm('sinusoid');  
mstruct.origin = [0 180 0];  
mstruct = defaultm(sinusoid(mstruct));
```

Having defined the map projection parameters, you can project the latitude and longitude vectors into the proper coordinates of the Sinusoidal projection and display the result using non-mapping, MATLAB commands.

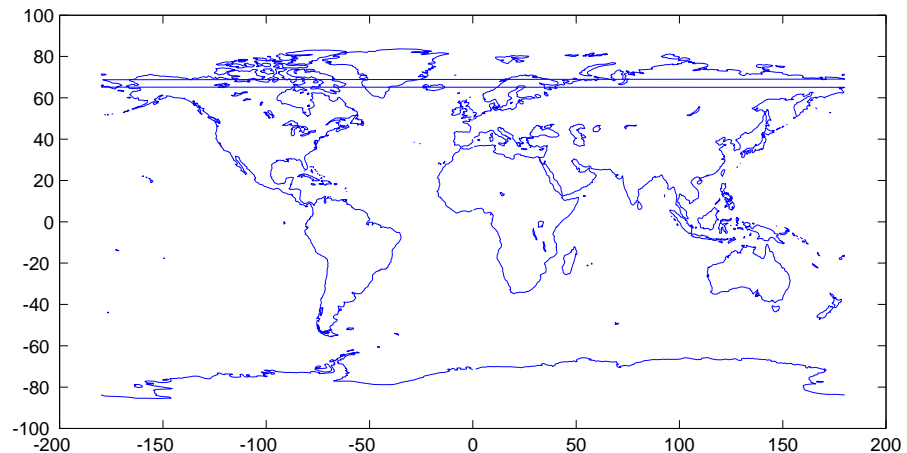
```
[x, y] = mfwdtran(mstruct, lat, long, [], 'line');
plot(x, y); axis equal
set(gca, 'XLim', [-3.5 3.5], 'YLim', [-2 2])
```



For the transformation function, an empty matrix is used to take the place of altitude data, and the string 'line' identifies the object type and allows the vector data to be clipped and trimmed. Notice that the International Date Line is now at the center of the map (180°), as specified earlier.

You can transform the projected x-y data back into the Greenwich geographic coordinates with the inverse transformation function. Plot the result:

```
[lat2, long2] = minvtran(mstruct, x, y);  
plot(long2, lat2); axis equal  
set(gca, 'XLim', [-200 200], 'YLim', [-100 100])
```



What happened? Recall that the original data extended beyond 180° longitude. In the projection transformation process, longitude data outside $[-180\ 180]$ degrees is projected back into this range since angles differing by 360° are geographically equivalent. The data from the inverse transformation process therefore jumps from 180° to -180° , as depicted by the straight horizontal lines in the figure above.

Summary Guide

Literally an infinite number of possible map projections can be constructed. The Mapping Toolbox provides 60 different map projections. To view all the projections in the toolbox, use the command `maps`.

Cartographers choose a map projection by determining which property distortions they want to minimize or eliminate, if any. They also determine which of the three projection types (cylindrical, conic, or azimuthal) best suits their purpose, or they may decide upon special desired features, such as straight rhumb lines or great circles, true direction, or some other interesting trait. The following table lists the available projections along with their properties. Detailed information on all map projections provided by the Mapping Toolbox can be found in the *Mapping Toolbox Reference Guide* under “Projections Reference.”

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Balthasart	bal thsrt	Cylindrical	•			
Behrmann	behrmann	Cylindrical	•			
Bolshoi Sovietskii Atlas Mira	bsam	Cylindrical				
Braun Perspective	braun	Cylindrical				
Cassini	cassi ni	Cylindrical			•	
Central	ccylin	Cylindrical				
Equal-Area Cylindrical	eqacylin	Cylindrical	•			
Equidistant Cylindrical	eqdcylin	Cylindrical			•	
Gall Isographic	giso	Cylindrical			•	
Gall Orthographic	gortho	Cylindrical	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Gall Stereographic	gstereo	Cylindrical				
Lambert Equal-Area Cylindrical	lambcyl n	Cylindrical	•			
Mercator	mercator	Cylindrical		•		1
Miller	mi l l e r	Cylindrical				
Plate Carree	pcarree	Cylindrical			•	
Trystan Edwards	trystan	Cylindrical	•			
Wetch	wetch	Cylindrical				
Apianus II	api anus	Pseudocylindrical				
Collignon	coll i g	Pseudocylindrical	•			
Craster Parabolic	craster	Pseudocylindrical	•			
Eckert I	eckert1	Pseudocylindrical				
Eckert II	eckert2	Pseudocylindrical	•			
Eckert III	eckert3	Pseudocylindrical				
Eckert IV	eckert4	Pseudocylindrical	•			
Eckert V	eckert5	Pseudocylindrical				
Eckert VI	eckert6	Pseudocylindrical	•			
Fournier	fourni er	Pseudocylindrical	•			
Goode Homolosine	goode	Pseudocylindrical	•			
Hatano Assymetrical Equal-Area	hat ano	Pseudocylindrical	•			

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Kavraisky V	kavrsky5	Pseudocylindrical	•			
Kavraisky VI	kavrsky6	Pseudocylindrical	•			
Loximuthal	loximuth	Pseudocylindrical				2
McBryde-Thomas Flat-Polar Parabolic	flatplr	Pseudocylindrical	•			
McBryde-Thomas Flat-Polar Quartic	flatplr	Pseudocylindrical	•			
McBryde-Thomas Flat-Polar Sinusoidal	flatplrs	Pseudocylindrical	•			
Mollweide	mollweid	Pseudocylindrical	•			
Putnins P5	putnins5	Pseudocylindrical				
Quartic Authalic	quartic	Pseudocylindrical	•			
Robinson	robinson	Pseudocylindrical				
Sinusoidal	sinusoid	Pseudocylindrical	•			
Tissot Modified Sinusoidal	modsi	Pseudocylindrical	•			
Wagner IV	wagner4	Pseudocylindrical	•			
Winkel I	winkel	Pseudocylindrical				
Albers Equal-Area Conic	eqaconic	Conic	•			
Equidistant Conic	eqdconic	Conic			•	
Lambert Conformal Conic	lambert	Conic		•		
Murdoch I Conic	murdoch1	Conic			•	3
Murdoch III Minimum Error Conic	murdoch3	Conic			•	3

Projection	Syntax	Type	Equal-Area	Conformal	Equidistant	Special Features
Bonne	bonne	Pseudoconic	•			
Werner	werner	Pseudoconic	•			
Polyconic	polycon	Polyconic				
Van Der Grinten I	vgrint1	Polyconic				
Breusing Harmonic Mean	breusing	Azimuthal				
Equidistant Azimuthal	eqdazi m	Azimuthal			•	
Gnomonic	gnomoni c	Azimuthal				4
Lambert Azimuthal Equal-Area	eqaazi m	Azimuthal	•			
Orthographic	ortho	Azimuthal				
Stereographic	stereo	Azimuthal		•		5
Wiechel	wichel	Pseudoazimuthal	•			
Globe	gl o b e	Spherical	•	•	•	6

- 1. Straight rhumb lines.
- 2. Rhumb lines from central point are straight, true to scale, and correct in azimuth.
- 3. Correct total area.
- 4. Straight line great circles.
- 5. Great and small circles appear as circles or lines.
- 6. Three-dimensional display.

Mapping Applications

Introduction to Geostatistics and Navigation.....	5-2
Geostatistics	5-3
Geographic Means	5-3
Geographic Standard Deviation	5-5
Equal-Areas in Geostatistics.....	5-7
Geographically Filtering Datasets	5-11
Navigation	5-12
Conventions for Navigational Functions	5-12
Fixing Position	5-13
Planning	5-27
Track Laydown – Displaying Navigational Tracks	5-28
Dead Reckoning	5-28
Time Zones	5-36
Time Notation	5-38

Introduction to Geostatistics and Navigation

The Mapping Toolbox provides a set of functions for applying mapping computations to geostatistical analysis and navigation. The geostatistics functions are necessary for proper analysis of geographical data as opposed to simple Cartesian (x-y) data. The navigation functions provide such useful navigational tasks as establishing position and planning routes or tracks.

Geostatistics

Most statistical functions implicitly assume that the data resides in a Cartesian coordinate system. Simply using geographic data in a Cartesian manner can give statistically inappropriate results. While the Cartesian assumption may be insignificant for small geographic regions, for larger areas it can lead to incorrect conclusions due to faulty distance measures, faulty area assumptions, or both. The Mapping Toolbox provides several functions for analyzing geographic data statistically.

Geographic Means

Consider the problem of calculating the mean position of a collection of geographic points. You might be inclined to simply take the arithmetical mean of the latitudes and longitudes using the standard MATLAB `mean` command, but this could be very misleading.

Take two points at the same latitude, 180° apart in longitude, for example $(30^\circ\text{N}, 90^\circ\text{W})$ and $(30^\circ\text{N}, 90^\circ\text{E})$. The “mean” latitude is $(30+30)/2=30$, which seems right. However, the “mean” longitude must be $(90+(-90))/2=0$. From the point of view at the Prime Meridian this seems fine, too. At the 0° meridian, each point has the same longitude difference from the mean. What about the point of view at the Date Line? At the 180° meridian, each point also has the same longitude difference. Why isn't 180° the mean longitude?

This problem is further complicated when some points are at different latitudes. Remember, a degree of longitude at the Arctic Circle covers a much smaller distance than a degree at the Equator.

In fact, in the first example, is 30°N the right mean latitude, either? The mean position of two points should be equidistant from those two points, and further, it should be the equidistant point that minimizes this distance. Is $(30^\circ\text{N}, 0^\circ)$ a reasonable mean point?

```
dist1 = distance(30, 90, 30, 0)
dist1 =
    75.5225
dist2 = distance(30, -90, 30, 0)
dist2 =
    75.5225
```

Here is another point, call it (lat, lon), that is also equidistant from both points, and the distance is much shorter:

```
dist1 = distance(30, 90, lat, lon)
dist1 =
    60.0000
dist2 = distance(30, -90, lat, lon)
dist2 =
    60.0000
```

What is this mystery point? The lat is 90°N, and any lon will do. The North Pole is the true geographic mean of these two points. Note that the great circle containing both points also runs through the North Pole, and a great circle represents the shortest path between two points.

The Mapping Toolbox includes the function `meanm` to determine the geographic mean of any number of points. This is accomplished through the three-dimensional vector addition of all the points. For example, try the following:

```
lats = [30 30];
longs = [-90 90];
[latbar, longbar] = meanm(lats, longs)
latbar =
    90
longbar =
    0
```

This is the answer we now expect. This geographic mean can result in one oddity; if the vectors all cancel each other, the mean is the center of the planet. In this case, the returned mean point is (NaN,NaN) and a warning is displayed. This phenomenon is highly improbable in “real” data, but can be easily constructed. For example, it occurs when all the points are equally spaced along a great circle. Try taking the geographic mean of (0°,0°), (0°,120°), and (0°,240°), which trisect the Equator.

Geographic Standard Deviation

As you might now expect, the Cartesian definition of standard deviation provided in the standard MATLAB function `std` is also inappropriate for most geographic data. Depending upon your purpose, you may want to use the separate geographic deviations for latitude and longitude provided by the function `stdm`, or the single standard distance provided in `stdist`. Both methods measure the deviation of points from the mean position calculated by `meanm`.

The Meaning of `stdm`

The `stdm` function handles the latitude and longitude deviations separately.

```
[latstd, lonstd] = stdm(lat, lon)
```

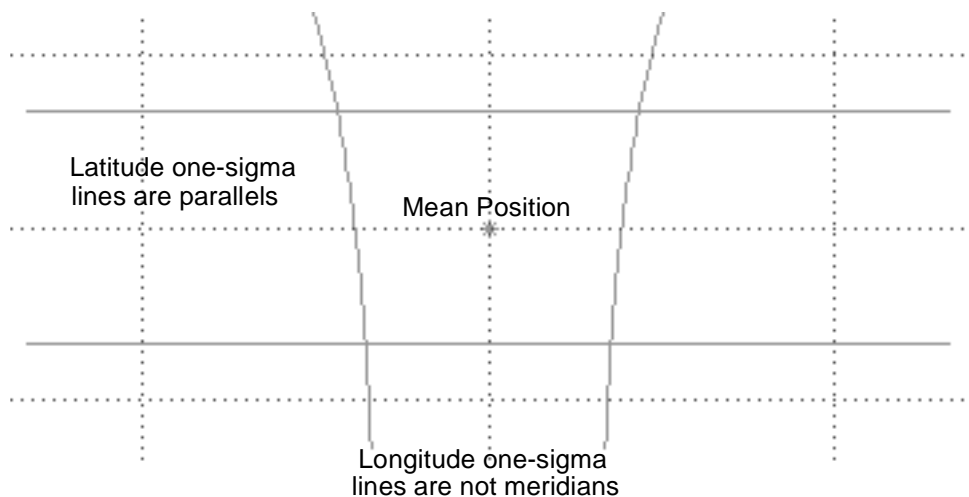
The function returns two deviations, one for latitudes and one for longitudes.

Latitude deviation is a straightforward standard deviation calculation from the mean latitude (mean parallel) returned by `meanm`. This is a reasonable measure for most cases, since a degree of latitude always has the same arc length.

Longitude deviation is another matter. Simple calculations based on sum-of-squares angular deviation from the mean longitude (mean meridian) are misleading. The arc length represented by a degree of longitude at extreme latitudes is significantly smaller than that at low latitudes.

The term *departure* is used to represent the arc length distance along a parallel of a point from a given meridian. For example, assuming a spherical planet, the departure of a degree of longitude at the Equator is a degree of arc length, but the departure of a degree of longitude at a latitude of 60° is one-half a degree of arc length. The `stdm` function calculates a sum-of-squares departure deviation from the mean meridian.

If you want to plot the one-sigma lines for `stdm`, the latitude sigma lines are parallels. However, the longitude sigma lines are not meridians; they are lines of constant departure from the mean parallel.



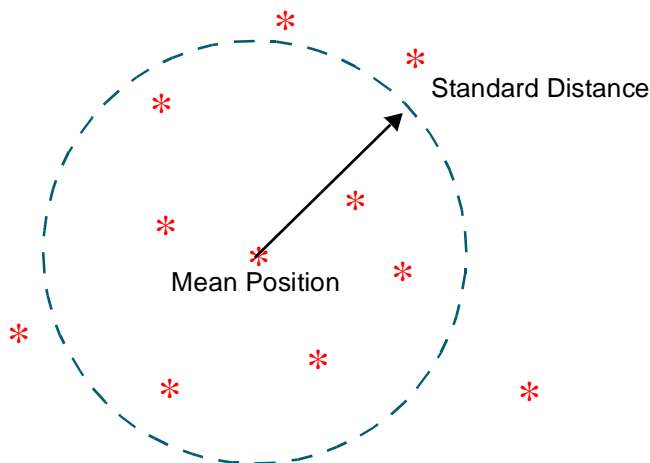
This handling of deviation has its problems. For example, its dependence upon the logic of the coordinate system can cause it to break down near the poles. For this reason, the standard distance provided by `stdist` is often a better measure of deviation. The `stdm` handling is useful for many applications, especially when the data is not global. For instance, these potential difficulties would not be a danger for data points confined to the country of Mexico.

The Meaning of `stdist`

The standard distance of geographic data is a measure of the dispersion of the data in terms of its distance from the geographic mean. Among its advantages are its applicability anywhere on the globe and its single value:

$$\text{dist} = \text{stdist}(\text{lat}, \text{lon})$$

In short, the standard distance is the average, norm, or “cubic-norm” of the distances of the data points in a great circle sense from the mean position. It is probably a superior measure to the two deviations returned by `st dm` except when a particularly latitude- or longitude-dependent feature is under examination



Equal-Areas in Geostatistics

A common error in applying two-dimensional statistics to geographic data lies in ignoring equal-area treatment. It is often necessary to “bin-up” data to statistically analyze it. In a Cartesian plane, this is easily done by dividing the space into equal x - y squares. The geographic equivalent of this is to bin up the data in equal latitude-longitude “squares.” Since such “squares” at high latitudes cover smaller areas than their low-latitude counterparts, the observations in these regions are underemphasized. The result can be conclusions that are biased towards the Equator.

Geographic Histograms

The geographic histogram command `hi str` allows you to display “binned-up” geographic observations. The `hi str` command results in equirectangular binning. Each bin has the same angular measurement in both latitude and longitude, with a default measurement of 1 degree. The center latitudes and longitudes of the bins are returned, as well as the number of observations per bin:

```
[bin lat, bin lon, num] = hi str(lats, lons)
```

As previously noted, these equirectangular bins result in counting bias towards the Equator. Here is a display of the one-degree-by-one-degree binning of approximately 5,000 random data points in Russia. The relative size of the circles indicates the number of observations per bin:



This is a portion of the whole map, displayed in an equal-area Bonne projection. The first step in creating data displays without area bias is to choose an equal-area projection. The proportionally sized symbols are a result of the specialized display function `scatterm`.

The area bias can be eliminated by adding a fourth output argument to `hist`, which will be used to weight each bin's observation by that bin's area:

```
[binlat, binlon, num, wnum] = hist(lats, lons)
```

The fourth output is the weighted observation count. Each bin's observation count is divided by its normalized area. Therefore, a high-latitude bin will have a larger weighted number than a low-latitude bin with the same number of actual observations. The same data and bins look much different when they are area-weighted:



Notice that there are larger symbols to the north in this display. The previous display suggested that the data was relatively uniformly distributed. When equal-area considerations are included, it is clear that the data is skewed to the north. In fact, the data used is northerly skewed, but a simple equirectangular handling failed to demonstrate this.

The `hist` command, therefore, does provide for the display of area-weighted data. However, the actual bins used are of varying areas. Remember, the one-degree-by-one-degree bin near a pole is much smaller than its counterpart near the Equator.

The `hist` command provides for actual equal-area bins.

Converting to an Equal-Area Coordinate System

Finally, on the topic of equal-area data handling, the actual data itself can be converted to an equal-area coordinate system for analysis with other statistical functions. It is easy to convert a collection of geographic latitude-longitude points to an equal-area x - y Cartesian coordinate system. The `grn2eqa` command applies the same transformation used in calculating the Equal-Area Cylindrical projection:

$$[x, y] = \text{grn2eqa}(lat, lon)$$

For each Greenwich lat - lon pair, an equal-area x - y is returned. The variables x and y can then be operated on under the equal-area assumption, using a variety of two-dimensional statistical techniques. Tools for such analysis can be found in the Statistics Toolbox and elsewhere. The results can then be converted back to Greenwich coordinates using the `eqa2grn` command:

$$[lat, lon] = \text{eqa2grn}(x, y)$$

Remember, when converting back and forth between systems, latitude corresponds to y and longitude corresponds to x .

Geographically Filtering Datasets

Often, a set of data will contain unwanted data mixed in with the desired values. For example, your data might include points for the entire United States, but you only want to work with those points falling in Alabama, or perhaps the dataset is “untidy” – out of 4,000 points, you notice that 3 or 4 obviously fall outside of reality (for example, one of your city-points is in the middle of the ocean). It can be quite a chore to look at each dataset element individually. Perhaps selecting a portion of the data is part of your analysis.

The `filter` command works with a matrix map to filter a vector dataset. The form is of the following:

```
[flats, flons] = filter(lats, lons, map, maplegend, allowed)
```

In short, each location defined by `lats` and `lons` is compared to the value at that point in `map`. If the value is `allowed`, that point is included in `flats` and `flons`.

The map might be a politically indexed map, and the allowed values might be the code or codes corresponding to the states or countries desired (e.g., Alabama). The map might also be a valued map, or a logical condition thereon, and the allowed value might be 1 for “true.” Here’s what an example might look like using the `topo` map:

```
[flats, flons] = filter(lats, lons, topo>0, toplegend, 1)
```

The result would be those points corresponding to land.

Navigation

One field that makes extensive use of geographic information is navigational science and practice. The Mapping Toolbox includes a variety of specialized functions for navigation.

The practice of navigation includes a variety of tasks for the operation of both watercraft and aircraft. One task is establishing position, using known, fixed landmarks (piloting), employing the stars, Sun, and Moon (celestial navigation), utilizing technological fixing systems (radio and satellite navigation, including GPS), or deducing net movement from a past known position (dead reckoning).

Another navigational task involves planning a voyage or flight, which includes determining a short route (great circle approximation), weather avoidance (optimal track routing), and setting out a plan of intended movement (track laydown).

The Mapping Toolbox contains functions for some of these navigational activities.

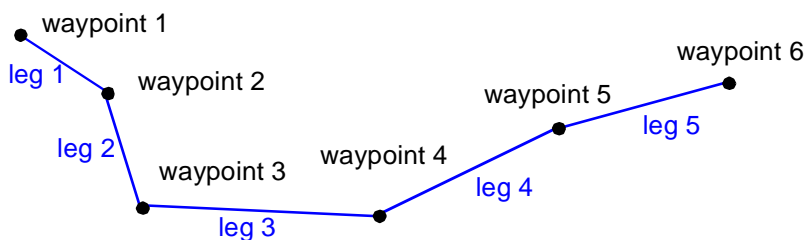
Conventions for Navigational Functions

Units

The Mapping Toolbox is, in general, very flexible in allowing a variety of angular and distance measurement units. To make the strictly navigational functions easy to handle and to conform to common navigational practice, *for these specific functions*, angles are always in degrees, distances are always in nautical miles, and speeds are always in knots (nautical miles per hour). The functions for which these conventions hold are `dreckon`, `gcwaypts`, `legs`, and `navfix`. Related functions that do not carry this restriction include `rhxrh`, `scxsc`, `gcxgc`, `gcxsc`, `track`, `timezone`, and `crossfix`, because of their potential for application outside of navigation.

Navigational Track Format

Navigational track format requires column-vector variables for the latitudes and longitudes of track waypoints. A *waypoint* is a point through which a track passes, usually corresponding to a course (or speed) change. Navigational tracks are made up of the line segments connecting these waypoints, which are called *legs*. In this format, therefore, n legs are described using $n+1$ waypoints, because an endpoint for the final leg must be defined.



Here, five track legs require six waypoints. In navigational track format, the waypoints would be represented by two 6-by-1 vectors, one for the latitudes and one for the longitudes.

Fixing Position

The fundamental objective of navigation is to determine at a given moment how to proceed to your destination, avoiding hazards on the way. The first step in accomplishing this is to establish your current position. Early sailors kept within sight of land to facilitate this. Today, navigation within sight (or radar range) of land is called *piloting*. Positions are fixed by correlating the bearings and/or ranges of landmarks. In real-life piloting, all sighting bearings are treated as rhumb lines, while in fact they are actually great circles.

Over the distances involved with visual sightings (up to 20 or 30 nautical miles), this assumption causes no measurable error and it provides the significant advantage of allowing the navigator to plot all bearings as straight lines on a Mercator projection.

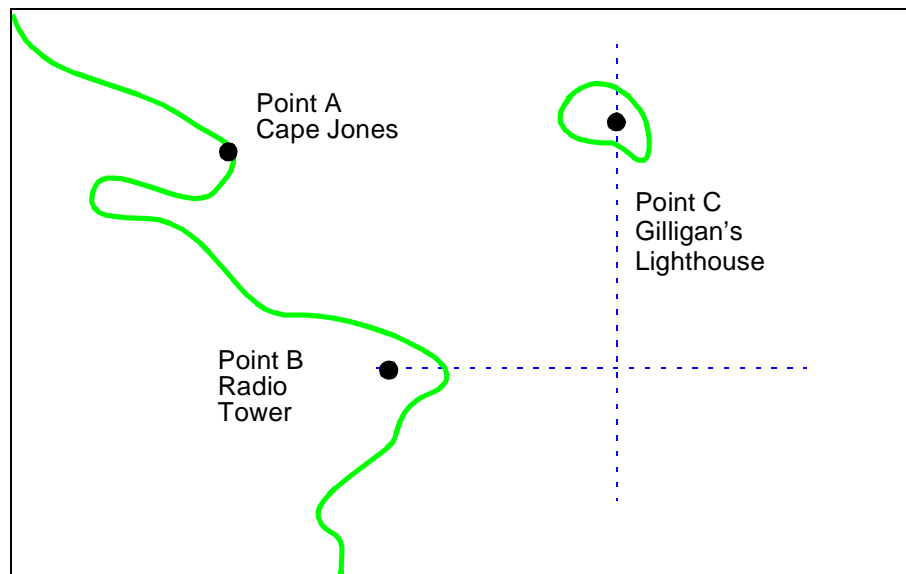
The Mercator was designed exactly for this purpose. Range circles, which might be determined with a radar, are assumed to plot as true circles on a Mercator chart. This allows the navigator to manually draw the range arc with a compass.

These assumptions also lead to computationally efficient methods for fixing positions with a computer. The Mapping Toolbox includes the `navfix` function, which mimics the manual plotting and fixing process using these assumptions.

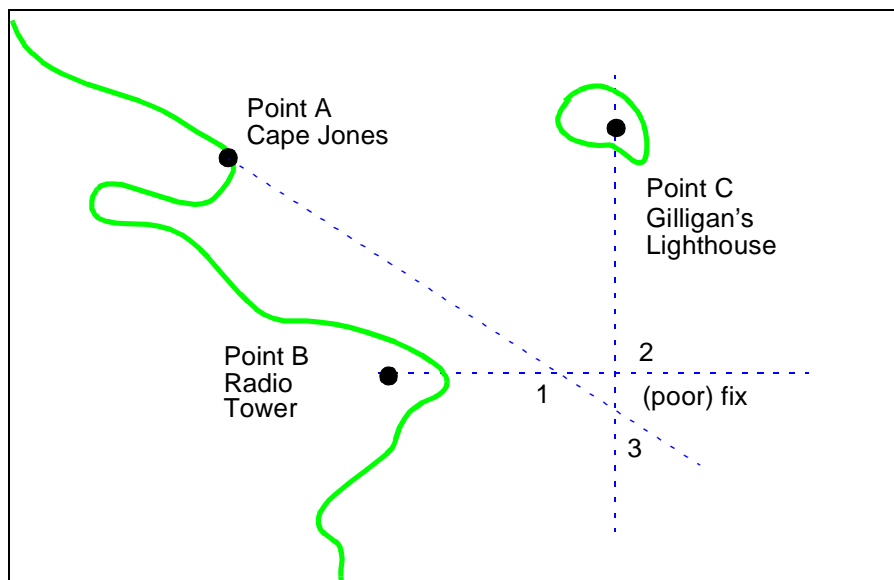
To obtain a good navigational fix, your relationship to at least three known points is considered necessary. A questionable or poor fix can be obtained with two known points.

Some Possible Situations

In this imaginary coastal region, you take a visual bearing on the radio tower of 270° . At the same time, Gilligan's Lighthouse bears 0° . If you plot a 90° - 270° line through the radio tower, and a 0° - 180° line through the lighthouse on your Mercator chart, the point at which the lines cross is a fix. Since you have used only two lines, however, its quality is questionable.



But wait; your port lookout says he took a bearing on Cape Jones of 300° . If that line exactly crosses the point of intersection of the first two lines, you will have a perfect fix.

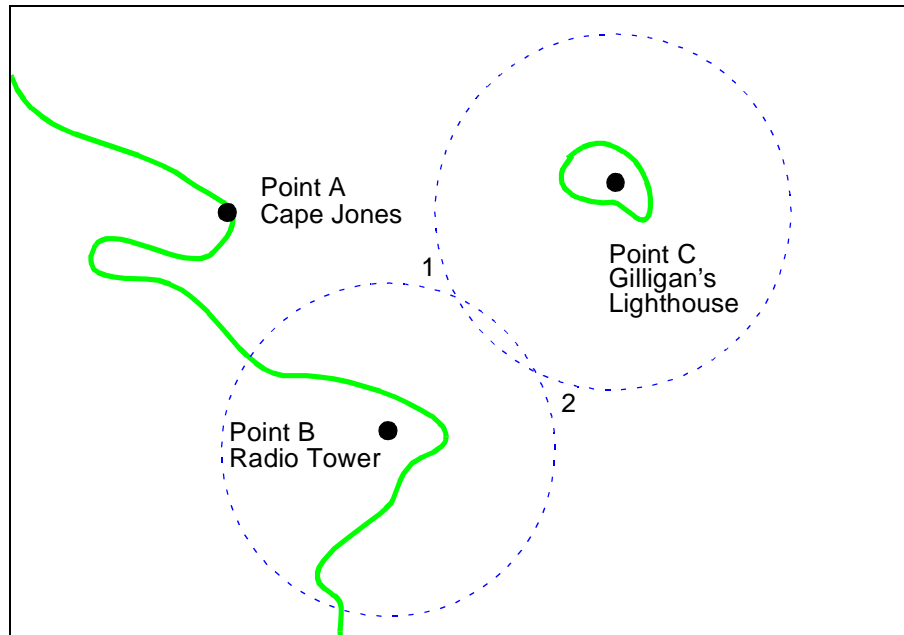


Whoops. What happened? Is your lookout in error? Possibly, but perhaps one or both of *your* bearings was slightly in error. This happens all the time. Which point – 1, 2, or 3 – is correct? As far as you know, they are all equally valid.

In practice, the little triangle is plotted, and the fix position is taken as either the center of the triangle or the vertex closest to a danger (like shoal water). If the triangle is large, the quality is reported as “poor,” or even as “no fix.” If a fourth line of bearing is available, it can be plotted to try to resolve the ambiguity. When all three lines appear to cross at exactly the same point, the quality is reported as “excellent” or “perfect.”

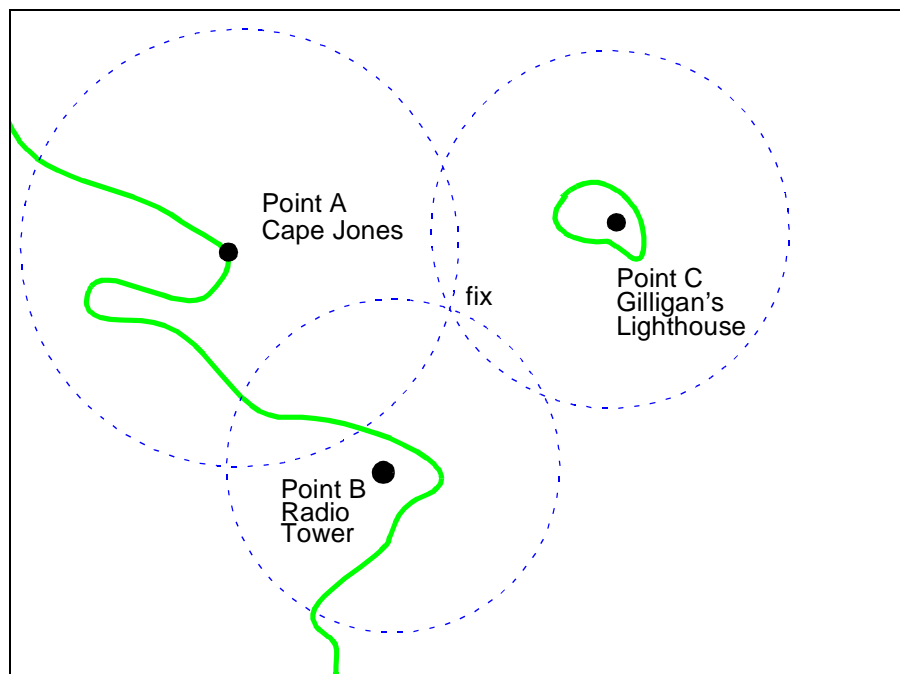
Notice that three lines resulted in three intersection points. Four lines would return six intersection points. This is a case of combinatorial counting. Each intersection corresponds to choosing two lines to intersect from among n lines, so the total number of intersection points will be n -choose-2.

The next time you traverse these straits, it is a very foggy morning. You can't see any landmarks, but luckily, your navigational radar is operating. Each of these landmarks has a good radar signature, so you're not worried. You get a range from the radio tower of 14 nautical miles and a range from the lighthouse of 15 nautical miles.



Now what? You took ranges from only two objects, and yet you have two possible positions. This ambiguity arises from the fact that circles can intersect twice.

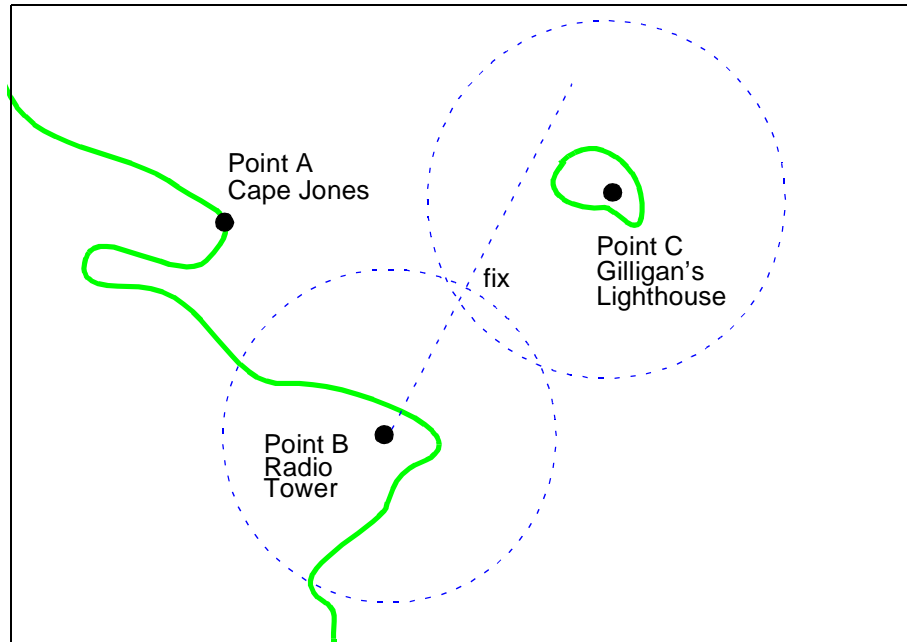
Luckily, your radar watch reports that he has Cape Jones at 18 nautical miles. This should resolve everything.



You were lucky this time. The third range resolved the ambiguity and gave you an excellent fix. Three intersections practically coincide. Sometimes the ambiguity is resolved, but the fix is still poor because the three closest intersections form a sort of circular triangle.

Sometimes the third range only adds to the confusion, either by bisecting the original two choices, or by failing to intersect one or both of the other arcs at all. In general, when n arcs are used, $2 \times (n \text{ choose } 2)$ possible intersections result. In this example, it is easy to tell which ones are “right.”

Bearing lines and arcs can be combined. If instead of reporting a third range, your radar watch had reported a bearing from the radar tower of 20° , the ambiguity could have also been resolved. Note, however, that in practice, lines of bearing for navigational fixing should only be taken visually, except in desperation. A radar's beamwidth can be a degree or more, leading to uncertainty.



As you begin to wonder whether this manual plotting process could be automated, your first officer shows up on the bridge with a laptop and the Mapping Toolbox.

Using `navfix`

The `navfix` command can be used to determine the points of intersection between any number of lines and arcs. Be warned, however, that due to the combinatorial nature of this process, the computation time grows rapidly with the number of objects. To illustrate this function, assign positions to the landmarks. Point A, Cape Jones, is at (`latA,lonA`). Point B, the radio tower, is at (`latB,lonB`). Point C, Gilligan's Lighthouse, is at (`latC,lonC`).

For the bearing-lines-only example, the syntax is:

```
[latfix,lonfix] = navfix([latA latB latC], [lonA lonB lonC], ...
                        [300 270 0])
```

This defines the three points and their bearings as taken *from the ship*. The outputs would look something like this, with actual numbers of course:

```
latfix =
    latfix1      NaN      % A intersecting B
    latfix2      NaN      % A intersecting C
    latfix3      NaN      % B intersecting C
lonfix =
    lonfix1      NaN      % A intersecting B
    lonfix2      NaN      % A intersecting C
    lonfix3      NaN      % B intersecting C
```

Notice that these are two-column matrices. The second column consists of NaNs because it is used only for the two-intersection ambiguity associated with arcs.

For the range-arcs-only example, the syntax is:

```
[latfix,lonfix] = navfix([latA latB latC], [lonA lonB lonC], ...
                        [16 14 15], [0 0 0])
```

This defines the three points and their ranges as taken *from the ship*. The final argument indicates that the three cases are all ranges.

The outputs have the following form:

```
latfix =
    latfix11  latfix12          % A intersecting B
    latfix21  latfix22          % A intersecting C
    latfix31  latfix32          % B intersecting C
lonfix =
    lonfix11  lonfix12          % A intersecting B
    lonfix21  lonfix22          % A intersecting C
    lonfix31  lonfix32          % B intersecting C
```

Here, the second column is used, because each pair of arcs has two potential intersections.

For the bearings *and* ranges example, the syntax requires the final input to indicate which objects are lines of bearing (indicated with a 1) and which are range arcs (indicated with a 0):

```
[latfix, lonfix] = navfix([latB latB latC], [lonB lonB lonC], ...
                          [20 14 15], [1 0 0])
```

The resulting output is mixed:

```
latfix =
    NaN          % Line B intersecting Arc B
    latfix21  latfix22          % Line B intersecting Arc C
    latfix31  latfix32          % Arc B intersecting Arc C
lonfix =
    NaN          % Line B intersecting Arc B
    lonfix21  lonfix22          % Line B intersecting Arc C
    lonfix31  lonfix32          % Arc B intersecting Arc C
```

Only one intersection is returned for the line from B with the arc about B, since the line originates inside the circle and intersects it once. The same line intersects the other circle twice, and hence it returns two points. The two circles taken together also return two points.

Usually, you have an idea as to where you are before you take the fix. For example, you might have a dead reckoning position for the time of the fix (*see below*). If you provide `navfix` with this estimated position, it will choose from each pair of ambiguous intersections the point closest to the estimate. Here's what it might look like:

```
[latfix, lonfix] = navfix([latB latB latC], [lonB lonB lonC], ...
                        [20 14 15], [1 0 0], drlat, drlon)

latfix =
    latfix11           % the only point
    latfix21           % the closer point
    latfix31           % the closer point
lonfix =
    lonfix11           % the only point
    lonfix21           % the closer point
    lonfix31           % the closer point
```

A Numerical Example of Using `navfix`

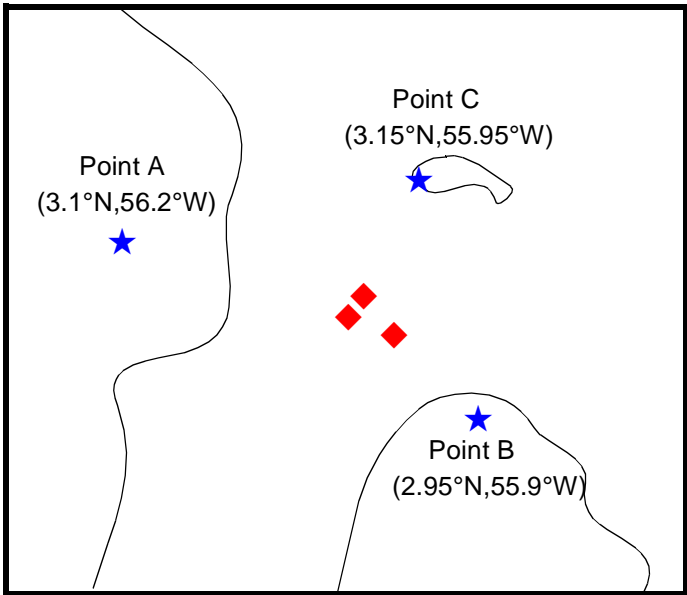
For a numerical example, define some specific points in the middle of the Atlantic Ocean. These are strictly arbitrary; perhaps they correspond to points in Atlantis:

```
lata = 3.1;  lona = -56.2;
latb = 2.95; lonb = -55.9;
latc = 3.15; lonc = -55.95;
```

Plot them on a Mercator projection:

```
axesm('MapProjection', 'mercator', 'Frame', 'on', ...
      'MapLatLimit', [2.8 3.3], 'MapLonLimit', [-55.8 -56.3])
plotm([lata latb latc], [lona lonb lonc], ...
      'LineStyle', 'none', 'Marker', 'pentagram', ...
      'MarkerEdgeColor', 'b', 'MarkerFaceColor', 'b', ...
      'MarkerSize', 12)
```

Here's what it looks like (the labeling and imaginary coastlines are added after the fact for illustration):



Take three visual bearings: Point A bears 289°, Point B bears 135°, and Point C bears 026.5°. Calculate the intersections:

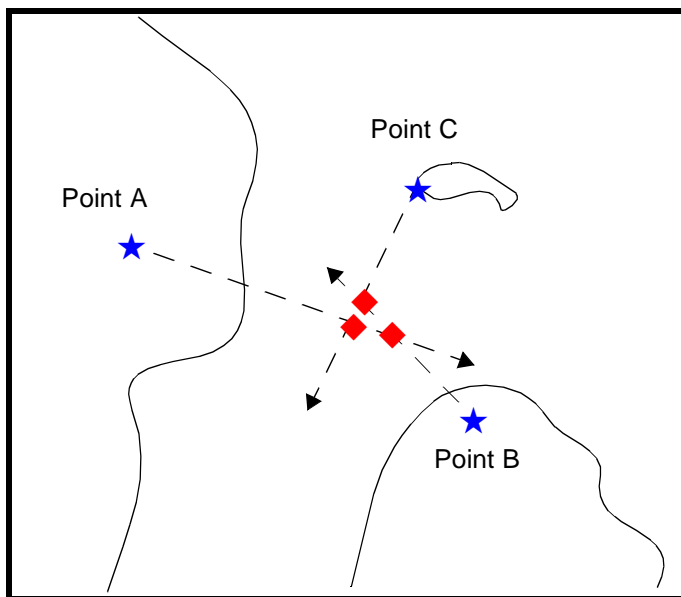
```
[newlat, newlong] = navfix([lata latb latc], [lona lonb lonc], ...
                           [289 135 26.5], [1 1 1])

newlat =
    3.0214      NaN
    3.0340      NaN
    3.0499      NaN

newlong =
   -55.9715      NaN
   -56.0079      NaN
   -56.0000      NaN
```

Add the bearing lines and intersection points to the map:

```
plotm(newlat, newlong, 'LineStyle', 'none', ...
      'Marker', 'diamond', 'MarkerEdgeColor', 'r', ...
      'MarkerFaceColor', 'r', 'MarkerSize', 9)
```



Notice that each pair of objects results in only one intersection, since all are lines of bearing.

What if instead, you had ranges from the three points, A, B, and C, of 13nm, 9nm, and 7.5nm, respectively.

```
[newlat, newlong] = navfix([lata latb latc], [lona lonb lonc], ...
                          [13 9 7.5], [0 0 0])
```

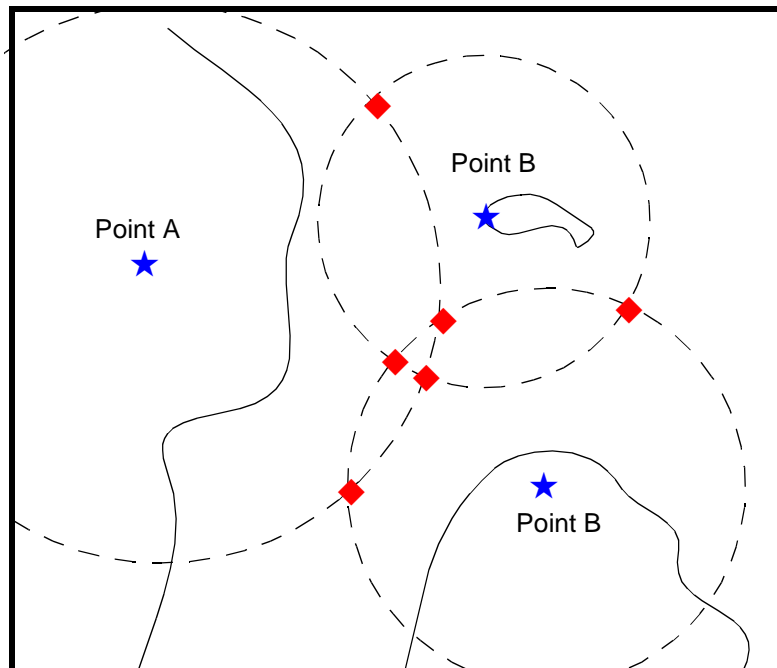
newlat =

```
3.0739    2.9434
3.2413    3.0329
3.0443    3.0880
```

newlong =

```
-55.9846  -56.0501
-56.0355  -55.9937
-56.0168  -55.8413
```

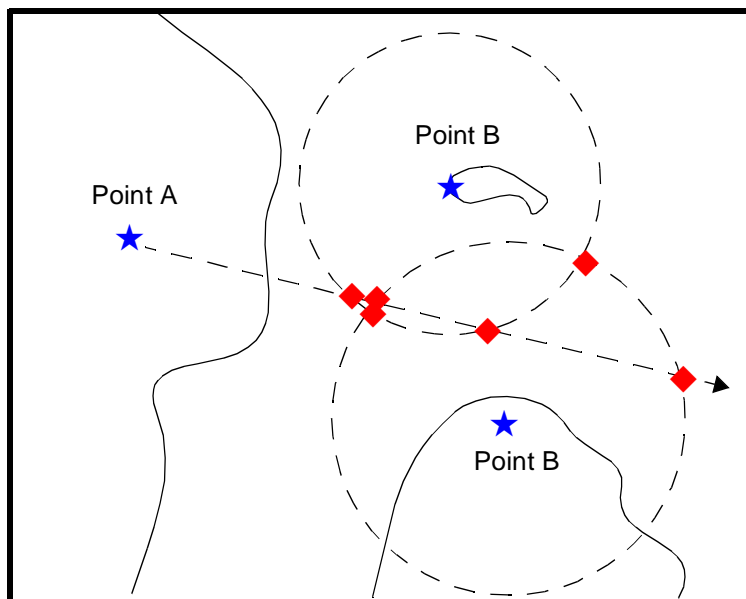
Here's what these points look like:



Three of these points look reasonable, three do not. What if, instead of a range from Point A, you had a bearing to it of 284° ?

```
[newlat, newlong] = navfix([lata latb latc], [lona lonb lonc], ...
                           [284 9 7.5], [1 0 0])
```

```
newlat =
    3.0526    2.9892
    3.0592    3.0295
    3.0443    3.0880
newlong =
   -56.0096  -55.7550
   -56.0360  -55.9168
   -56.0168  -55.8413
```

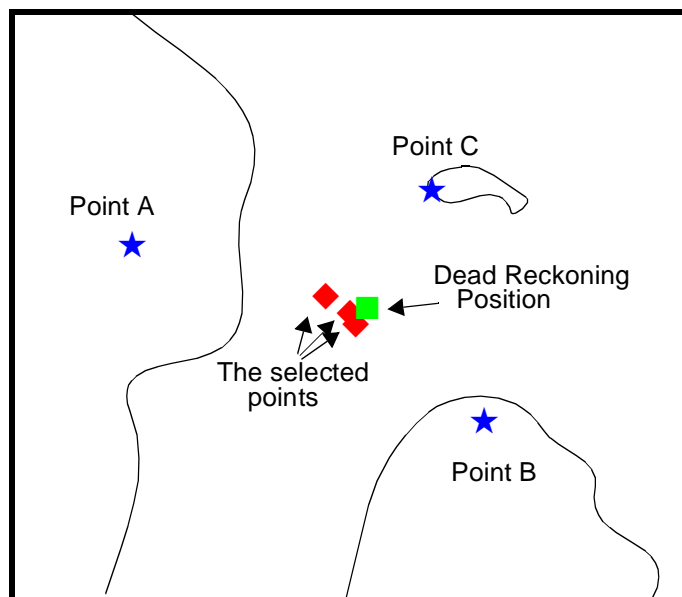


Again, visual inspection of the results indicates which three of the six possible points seem like “reasonable” positions. When using the dead reckoning position (3.05°N, 56.0°W), the closer, more reasonable candidate from each pair of intersecting objects is chosen:

```

drlat = 3.05; drlon = -56;
[newlat, newlong] = navfix([lata latb latc], [lona lonb lonc], ...
                           [284 9 7.5], [1 0 0], drlat, drlon)

newlat =
    3.0526
    3.0592
    3.0443
newlong =
   -56.0096
   -56.0360
   -56.0168
    
```



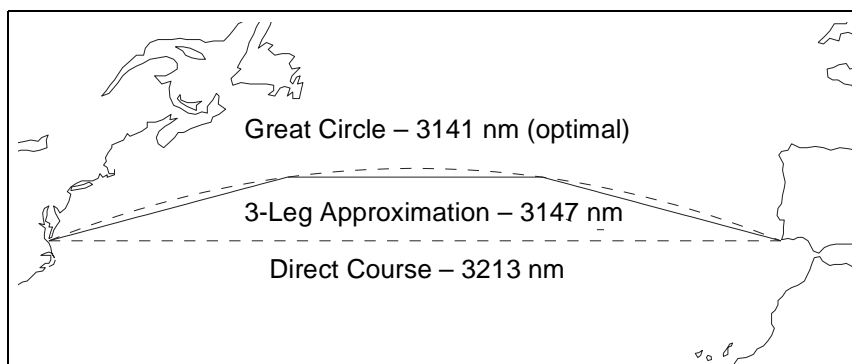
Planning

We know that the shortest path between two geographic points is a great circle. Sailors and aviators are interested in minimizing distance travelled, and hence delay. We also know that the rhumb line is a path of constant heading, the “natural” means of travelling. In general, to follow a great circle path, one would have to continuously alter course. This is impractical. However, a great circle path can be approximated by rhumb line segments so that the added distance is minor and the number of course changes minimal.

Surprisingly, very few rhumb line “track legs” are required to closely approximate the distance of the great circle path.

Consider the voyage from Norfolk, Virginia ($37^{\circ}\text{N}, 76^{\circ}\text{W}$), to Cape St. Vincent, Portugal ($37^{\circ}\text{N}, 9^{\circ}\text{W}$), one of the most heavily trafficked routes in the Atlantic. A due-east rhumb line track is 3,213 nautical miles in length, while the optimal great circle distance is 3,141 nautical miles.

Although the rhumb line path is only a little more than 2% longer, this is an additional 72 miles over the course of the trip. For a 12-knot tanker, this results in a 6-hour delay, and in shipping, time is money. If just three rhumb line segments are used to approximate the great circle, the total distance of the trip is 3,147 nautical miles. Our tanker would suffer only a half-hour delay over optimal.



The Mapping Toolbox provides the function `gcwaypts` to quickly calculate waypoints in navigation track format in order to approximate a great circle with rhumb line segments. The syntax is simple:

```
[latpts, lonpts] = gcwaypts(lat1, lon1, lat2, lon2, numlegs)
```

All of the inputs for this command are scalars. The `numlegs` input is the number of equal length legs desired, which is 10 by default. The outputs are column vectors representing waypoints in navigational track format. The size of each of these vectors will be `[(numlegs+1) 1]`. Here are the points for the example illustrated above:

```
[latpts, lonpts] = gcwaypts(37, -76, 37, -9, 3)
latpts =
    37.0000
    41.5076
    41.5076
    37.0000
lonpts =
   -76.0000
   -54.1777
   -30.8223
    -9.0000
```

These points represent waypoints along the great circle between which the approximating path follows rhumb lines. Four points are needed for three legs, because the final point at Cape St. Vincent must be included.

Track Laydown – Displaying Navigational Tracks

Navigational tracks are most useful when graphically displayed. Traditionally, the navigator identifies and plots waypoints on a Mercator projection and then connects them with a straightedge, which on this projection results in rhumb line tracks. In the previous example, waypoints were chosen to approximate a great circle route, but they may be selected for a variety of other reasons.

Let's say that after arriving at Cape St. Vincent, your tanker must traverse the Straits of Gibraltar and then travel on to Port Said, the northern terminus of the Suez Canal. On the scale of the Mediterranean Sea, following great circle paths is of little concern compared to ensuring that the many straits and passages are safely transited. The navigator selects appropriate waypoints and plots them.

To do this with the Mapping Toolbox, you can display a map axes with a Mercator projection, select appropriate map latitude and longitude limits to isolate the area of interest, plot coastline data, and interactively mouse-select the waypoints with the `inputm` command. The `track` command will generate points to connect these waypoints, which can then be displayed with `plotm`.

For illustration, assume that the waypoints are known (or were gathered using `inputm`). See the *Mapping Toolbox Reference Guide* or “Interacting with Displayed Maps” to learn about using `inputm`.

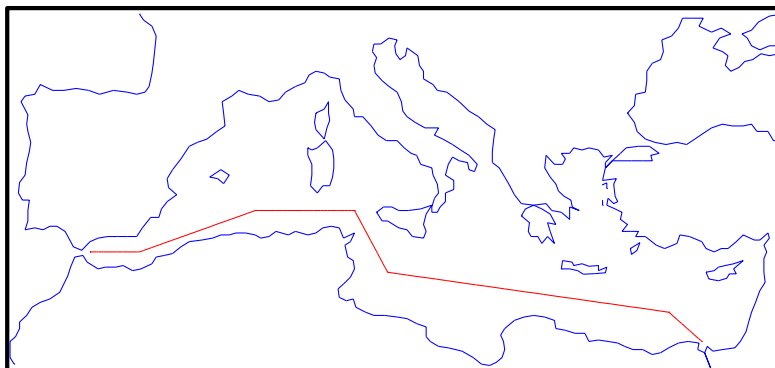
```

waypoints = [36 -5; 36 -2; 38 5; 38 11; 35 13; 33 30; 31.5 32]
waypoints =
    36.0000    -5.0000
    36.0000    -2.0000
    38.0000     5.0000
    38.0000    11.0000
    35.0000    13.0000
    33.0000    30.0000
    31.5000    32.0000

load coast
axesm('MapProjection','mercator',...
'MapLatLimit',[30 47], 'MapLonLimit', [-10 37])
framem
plotm(lat, long)

[ltrk, ltrk] = track(waypoints);
plotm(ltrk, ltrk, 'r')
```

Although these track segments are straight lines on the Mercator projection, they will be curves on others:



The segments of a track like this are called *legs*. Each of these legs can be described in terms of course and distance. The function `legs` will take the waypoints in navigational track format and return the courses and distances required for each leg. Remember, the order of the points in this format determines the direction of travel. Courses are therefore calculated from each waypoint to its successor, not vice-versa.

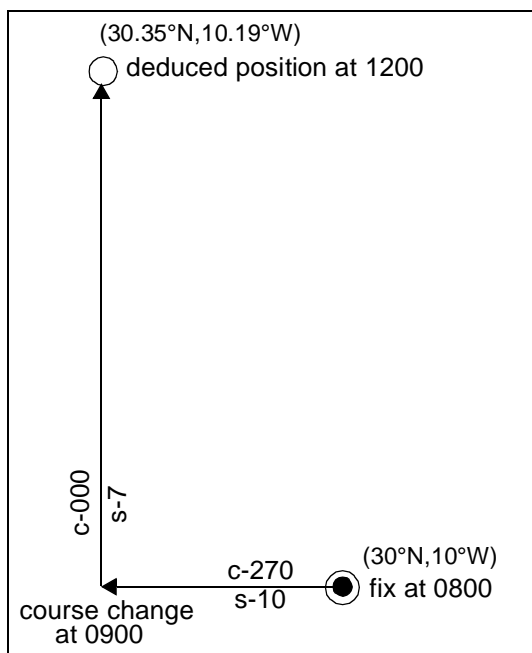
```
[courses, distances] = legs(waypoints)
courses =
    90.0000
    70.3132
    90.0000
   151.8186
    98.0776
   131.5684
distances =
   145.6231
   356.2117
   283.6839
   204.2073
   854.0092
   135.6415
```

Since this is a navigation function, the courses are all in degrees and the distances are in nautical miles. From these distances, speeds required to arrive at Port Said at a given time can be calculated. Southbound traffic is allowed to enter the canal only once per day, so this information might be economically significant, since unnecessarily high speeds can lead to high fuel costs.

Dead Reckoning

When sailors first ventured out of sight of land, they faced a daunting dilemma. How could they find their way home if they didn't know where they were? The practice of *dead reckoning* is an attempt to deal with this problem. The term is derived from "deduced reckoning."

Briefly, dead reckoning is vector addition plotted on a chart. For example, if you had a fix at $(30^{\circ}\text{N}, 10^{\circ}\text{W})$ at 0800, and you proceeded due west for 1 hour at 10 knots, and then you turned north and sailed for 3 hours at 7 knots, you should be at $(30.35^{\circ}\text{N}, 10.19^{\circ}\text{W})$ at 1200. All's well, ring eight bells on time, Mr. Bowditch, right?



However, Mr. Bowditch “shoots the sun” at local apparent noon and discovers that the ship’s latitude is actually 30.29°N . What’s worse, he lives before the invention of a reliable chronometer, and so he cannot calculate his longitude at all from this sighting. What happened?

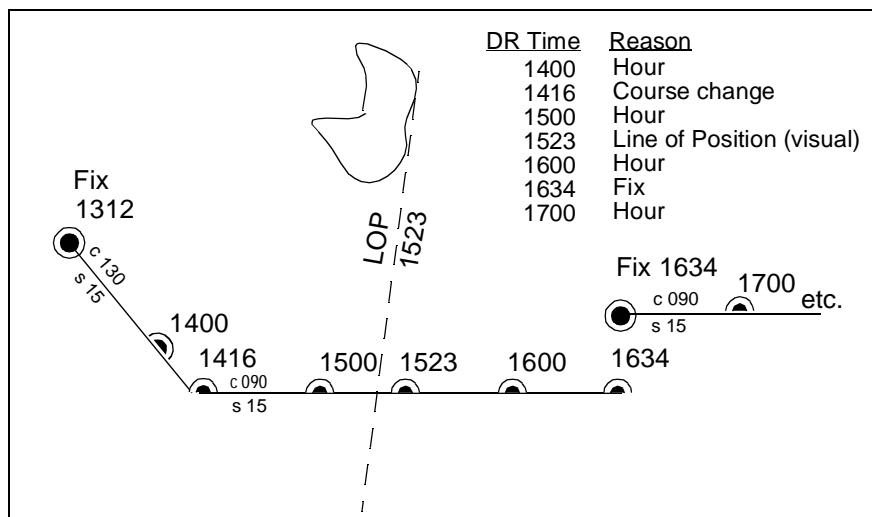
Leaving aside the difficulties in speed determination and the need to tack off course, even modern craft have to contend with winds and currents. However, despite these limitations, dead reckoning is still used for determining position between fixes and for forecasting future positions. This is because dead reckoning provides a certainty of assumptions that estimations of wind and current drift cannot.

When navigators establishes a fix from some source, be it from piloting, celestial, or satellite observations, they plot a dead reckoning (DR) track, which is a plot of the intended positions of the ship forward in time. In practice, dead reckoning is usually plotted for 3 hours in advance, or for the time period covered by the next three expected fixes. In open ocean conditions, hourly fixes are sufficient; in coastal pilotage, three-minute fixes are common.

Specific DR positions, which are sometimes called “DRs,” are plotted according to the “Rules of DR”:

- DR at every course change
- DR at every speed change
- DR every hour on the hour
- DR every time a fix or running fix is obtained
- DR three hours ahead or for the next three expected fixes
- DR for every line of position (LOP), either visual or celestial

For example, the navigator plots these DRs:



Notice that the 1523 DR does not coincide with the LOP at 1523. Although note is taken of this variance, one line is insufficient to calculate a new fix.

The Mapping Toolbox includes the function `dreckon`, which calculates the DR positions for a given set of courses and speeds. The function provides DR positions for the first three rules of dead reckoning. The approach is to provide a set of waypoints in navigational track format corresponding to the plan of intended movement.

The time of the initial waypoint, or fix, is also needed, as well as the speeds to be employed along each leg. Alternatively, a set of speeds and the times for which each speed will apply can be provided. `dreckon` will return the positions and times required of these DRs:

- `dreckon` calculates the times for positions of each course change, which will occur at the waypoints
- `dreckon` calculates the positions for each whole hour
- if times are provided for speed changes, `dreckon` calculates positions for these times, if they do not occur at course changes

Imagine you have a fix at midnight at the point (10°N,0°):

```
waypoints(1,:) = [10 0]; fixtime = 0;
```

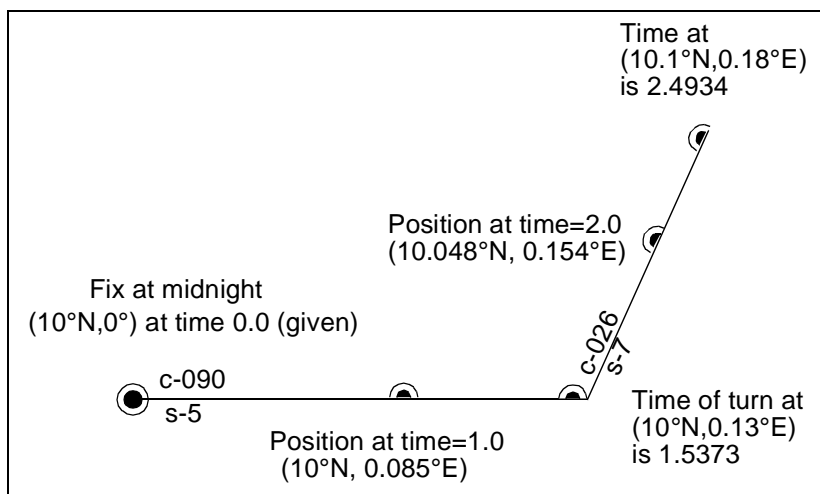
You intend to travel east and alter course at the point (10°N,0.13°E) and head for the point (10.1°N,0.18°E). On the first leg, you will travel at 5 knots, and on the second leg you will speed up to 7 knots.

```
waypoints(2,:) = [10 .13];
waypoints(3,:) = [10.1 .18];
speeds = [5; 7];
```

To determine the DR points and times for this plan, use `dreckon`:

```
[drlat, drlon, drtime] = dreckon(waypoints, fixtime, speeds);
[drlat drlon drtime]
ans =
    10.0000    0.0846    1.0000    % Position at 1 am
    10.0000    0.1301    1.5373    % Time of course change
    10.0484    0.1543    2.0000    % Position at 2 am
    10.1001    0.1801    2.4934    % Time at final waypoint
```

Here is an illustration of this track and its DR points:



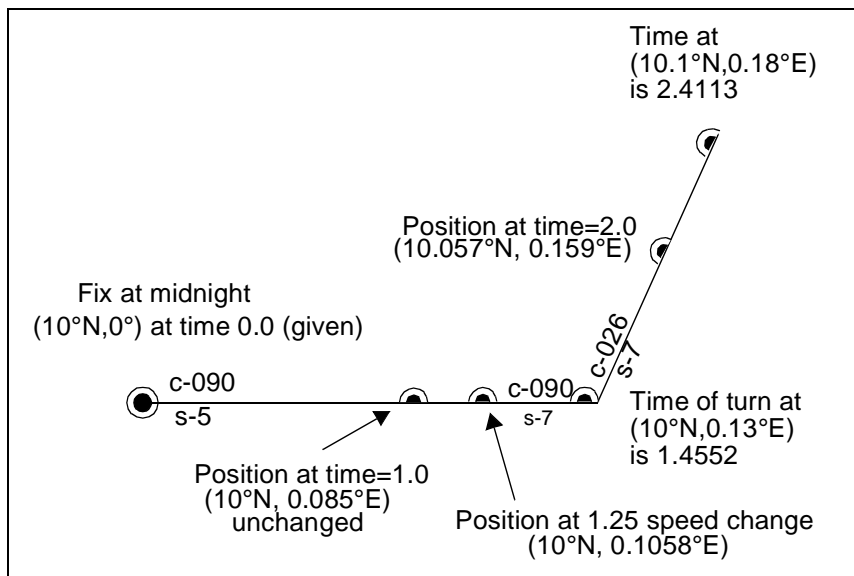
However, you would like to get to the final point a little earlier to make a rendezvous. You decide to recalculate your DRs based on speeding up to 7 knots a little earlier than planned. The first calculation tells you that you were going to increase speed at the turn, which would occur at a time 1.5373 hours after midnight, or 1:32 a.m. (at time 0132 in navigational time format). What time would you reach the rendezvous if you increased your speed to 7 knots at 1:15 a.m. (0115, or 1.25 hours after midnight)?

To indicate times for speed changes, another input is required, providing a time interval after the fix time at which *each ordered speed is to end*. The first speed, 5 knots, is to end 1.25 hours after midnight. Since you don't know when the rendezvous will be made under these circumstances, set the time for the second speed, 7 knots, to end at infinity. No DRs will be returned past the last waypoint.

```
spdtimes = [1.25; inf];
[drlat, drlon, drtime] = dreckon(waypoints, fixtime, ...
                                speeds, spdtimes);

[drlat, drlon, drtime]
ans =
    10.0000    0.0846    1.0000    % Position at 1 am
    10.0000    0.1058    1.2500    % Position at speed change
    10.0000    0.1301    1.4552    % Time of course change
    10.0570    0.1586    2.0000    % Position at 2 am
    10.1001    0.1801    2.4113    % Time at final waypoint
```

This following illustration shows the difference:

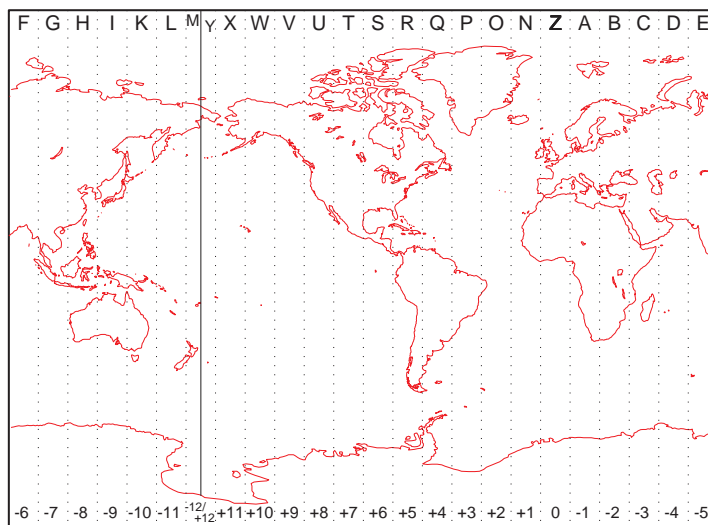


The times at planned positions after the speed change are a little earlier; the position at the known time (2 a.m) is a little farther along. With this plan, you will arrive at the rendezvous about 4-1/2 minutes earlier, so you may want to consider a more authoritative speed change.

Time Zones

Early in this manual, we gave an example of a time zone calculation. The `timezone` function returns a navigational time zone, that is, one based solely on longitude with no regard for statutory divisions. So, for example, Chicago, Illinois, lies in the statutory U.S. Central time zone, which has irregular boundaries devised for political or convenience reasons. However, from a navigational standpoint, Chicago's longitude places it in the "S" (Sierra) time zone. The zone's "description" is "+6", which indicates that 6 hours must be added to local time to get Greenwich, or "Z" (Zulu) time. So, if it is noon, standard time in Chicago, it is 12+6, or 6 p.m. at Greenwich.

Each navigational time zone is 15° of longitude in width and has a distinct description and designating letter. The exceptions to this are the two zones on either side of the Date Line, “M” and “Y” (Mike and Yankee). These zones are only 7-1/2° wide, since on one side of the Date Line, the description is +12, and on the other, it is -12. Navigational time zones are very important for celestial navigation calculations. Although the Mapping Toolbox does not contain any functions designed specifically for celestial navigation, a simple example can be devised.



It is possible with a sextant to determine “local apparent noon.” This is the moment when the Sun is at its zenith from your point of view. At the exact center longitude of a time zone, the phenomenon occurs exactly at noon, local time. Since the Sun traverses a 15° time zone in 1 hour, it crosses one degree every 4 minutes. So, if you observe local apparent noon at 11:54, you must be 1.5° east of your center longitude.

You must know what time zone you are in before you can even attempt a fix. This concept has been understood since the spherical nature of the Earth was first accepted, but early sailors had no ability to keep accurate time on ship, and so were unable to determine their longitude. The engineering exploitation of this scientific truism had to wait until the invention of the chronometer.

The `timezone` function is quite simple. It returns the description, `zd`, an integer for use in calculations, a string `zltr` of the zone designator, and a string fully naming the zone. For example, the information for a longitude 123°E is the following:

```
[zd, zltr, zone] = timezone(123)
zd =
    -8
zltr =
    H
zone =
    -8 H
```

Returning to our simple celestial navigation example, the center longitude of this zone is:

```
-(zd*15)
ans =
    120
```

This means that at our longitude, 123°E, we should experience local apparent noon at 11:48 a.m., 12 minutes early.

Time Notation

Navigational practice has its own peculiar notation for times. Time labels on navigation plots are always in a special format. Times are given in four digits, hours from 00 to 23 followed by minutes from 00 to 59. So, one minute before noon is 1159, or 1159Z or 1159Q, etc., based on time zone. Similarly, one minute after midnight is 0001. When more precision is required, the seconds are rounded to the nearest quarter minute and zero, one, two or three apostrophes are suffixed to the time, one for each 15-second block. So, 15 seconds before noon would be 1159''; 14 seconds before noon would have the exact same notation.

The Mapping Toolbox includes the function `time2str` that returns a string in a variety of formats corresponding to a given time. These strings can then be plotted on map displays as desired. Two other clock formats are also allowed – the 12-hour and the 24-hour digital clock readouts. Consider some string notations for the time 13.21 hours after midnight. The default 24-hour clock is:

```
time2str(13. 21)
ans =
13: 12: 36
```

The 12-hour clock reads:

```
time2str(13. 21, ' 12' )
ans =
01: 12: 36 PM
```

And the navigation format for this time is:

```
time2str(13. 21, ' nav' )
ans =
1312' '
```

Each of these can be rounded to the nearest minute with the third argument 'hm' (for hours-minutes – the default is 'hms'):

```
time2str(13. 21, ' nav' , ' hm' )
ans =
1313
```


GUI Tools

An Overview	6-2
The Complete GUI Environment: <code>maptool</code>	6-3
Starting <code>maptool</code>	6-3
The Menus	6-4
The Mouse Tool Buttons	6-4
Activating GUIs with <code>maptool</code> : A Working Example	6-5
Constructing Personalized Map Data with GUIs	6-31
Trimming an Existing Data Set	6-31
Encoding a Regular Surface Map	6-33
Creating a Personalized Colormap	6-41

An Overview

The Mapping Toolbox includes a number of graphical user interface (GUI) tools that provide an alternative to entering toolbox commands from the MATLAB command window. The Mapping Toolbox GUIs cover most, but not all, aspects of the mapping process.

The Mapping Toolbox GUI system is a collection of independent GUIs, most of which can be activated from the command line and/or from a map display window at any time during a MATLAB session. The Mapping Toolbox also provides `maptool`, a comprehensive tool that allows access to most of the toolbox's independent GUIs through a system of figure window menus and buttons.

Most of the Mapping Toolbox GUIs include a **Help** button used to enter help mode for that particular GUI. Once in help mode, pressing any enabled control will display the help text for that control in a separate help box. Pressing the **Done** button will close the help box and end help mode.

This chapter is a short tutorial on several of the Mapping Toolbox GUI tools. For a more detailed description of each tool, consult the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

The Complete GUI Environment: `maptool`

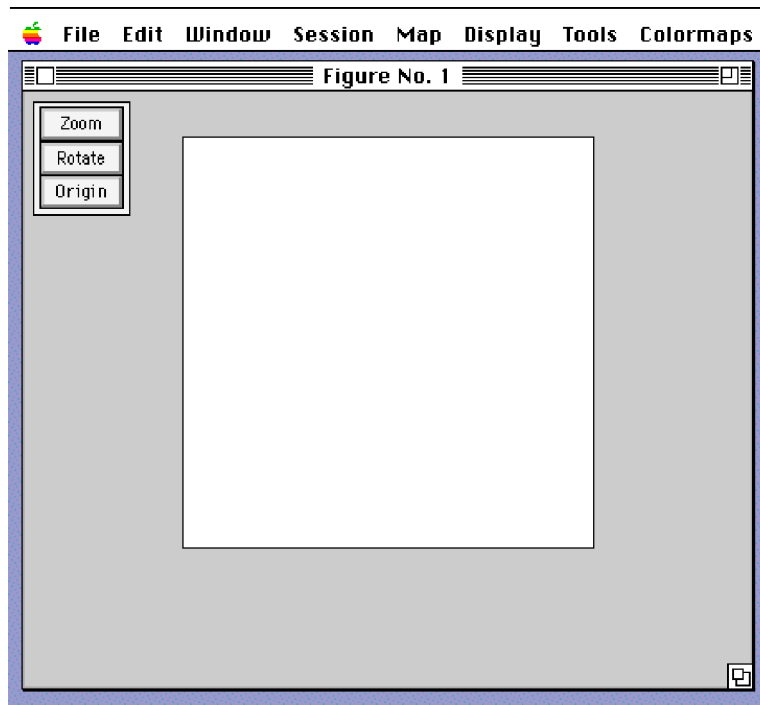
In this section, `maptool` is used to access many of the Mapping Toolbox GUIs. It should be noted, however, that most of the Mapping Toolbox GUIs can be activated independently at any time during a mapping session without the use of `maptool`. The “GUI Reference” section of the *Mapping Toolbox Reference Guide* provides details on the methods for activating each tool.

Starting `maptool`

You can start `maptool` in one of three ways:

```
maptool
maptool MapProjectionName
maptool (' MapPropertyName' , ' MapPropertyValue' )
```

Starting `maptool` creates a new figure window with a map axes and a set of pull-down menus:



All mapping functions activated through `maptool` will be associated with this map axes. If an initial map projection is not provided at the command line, the **Projection Control** dialog box will appear, with the Robinson projection as the default. In the upper left corner of the figure window is a set of three mouse tool buttons that allow interactive display manipulation.

The Menu

There are five pull-down menus associated with `maptool`.

- The **Session** menu manipulates the workspace and sets the figure window renderer for the `maptool` session.
- The **Map** menu command activates GUIs that are used to project objects onto the map axes.
- The **Display** menu edits map axes properties, manipulates display settings, and calculates and displays navigational tracks, small circles, and surface distances.
- The **Tools** menu provides a variety of miscellaneous mapping tools.
- The **Colormaps** menu allows for manipulation of the colormap for the current figure.

The Mouse Tool Buttons

There are three mouse tool buttons associated with `maptool`.

- The **Zoom** button toggles “Zoom” mode on and off. Zoom mode zooms in on a two-dimensional map display.
- The **Rotate** button toggles “Rotate 3-D” mode on and off. Rotate 3-D interactively rotates the view of a three-dimensional plot.
- The **Origin** button toggles “Origin” mode on and off. Origin mode interactively modifies the map origin.

For more details on the `maptool` menus and buttons, see the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

Activating GUIs with maptool: A Working Example

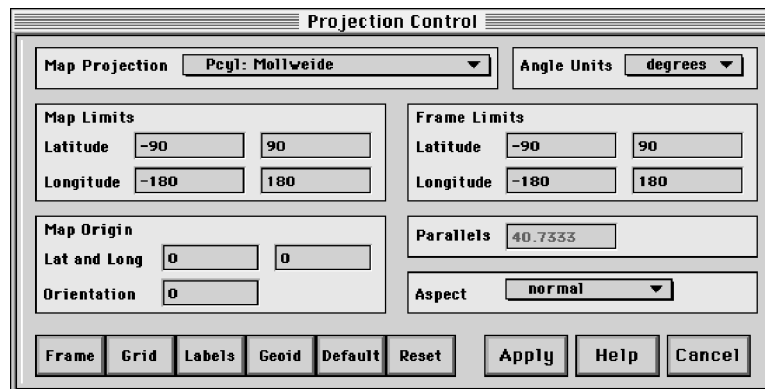
Suppose you are planning a cruise this summer, and you want to map out your travel itinerary. You will be flying out of Washington, D.C., to Barcelona, Spain, with a stopover in London, England. At Barcelona, you will board the cruise ship and sail to Monte Carlo, then on to Italy, sailing to Livorno, Rome, Naples, and finally through the Straits of Messina and up the Adriatic Sea to Venice. You will fly back home from Venice, with a stopover in Hamburg, Germany.

Setting Initial Map Properties

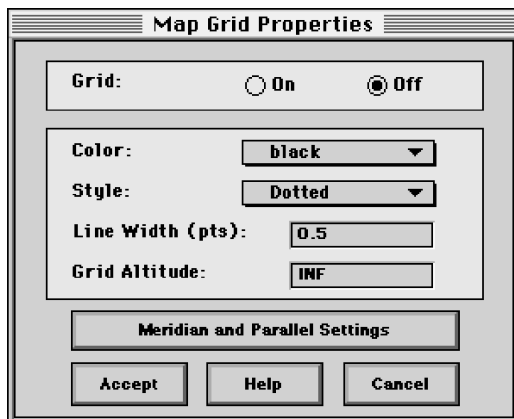
First, you would like to display your flight itinerary. From the MATLAB command prompt, type:

```
maptool
```

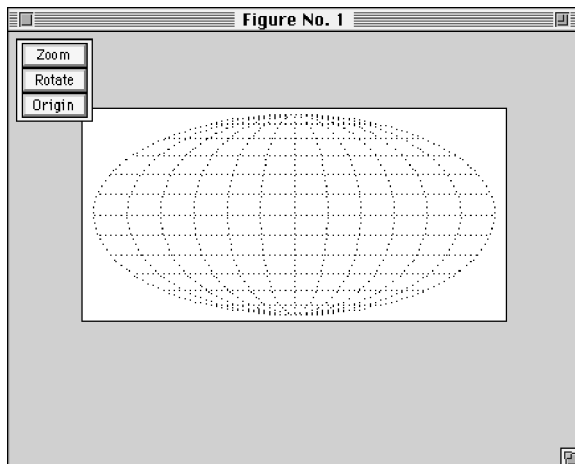
This creates a new figure window and brings up the **Projection Control** dialog box. Use the **Map Projection** pull-down menu to initiate a Mollweide projection:



Press the **Grid** button to activate the **Map Grid Properties** dialog box and turn the grid on:

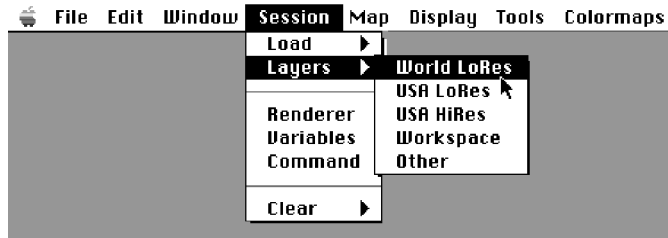


Press **Accept** to return to the **Projection Control** box, and press **Apply** to apply the settings to the map axes:

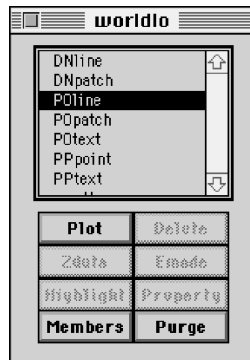


Plotting Layered Atlas Data

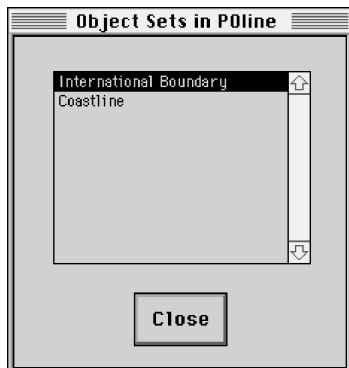
Select **Session⇒Layers⇒World LoRes** from the menu bar to activate the ml ayers GUI with the worl d l o workspace:



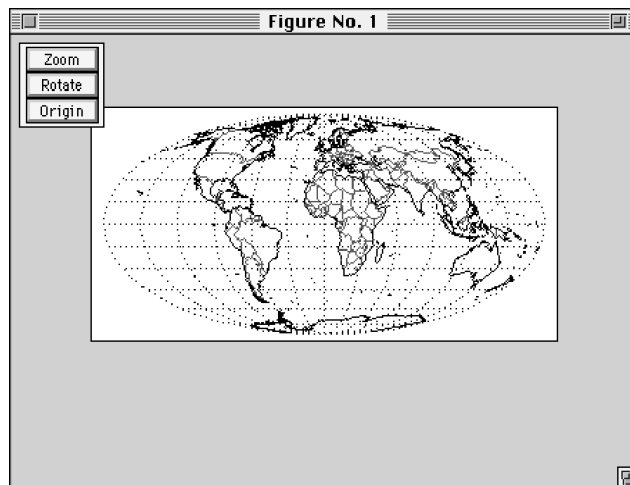
This brings up the **worldlo** dialog box, which lists all the structures in the worl d l o workspace:



Select **P0l i ne** in the list box, and press the **Plot** button. Pressing the **Members** button reveals that the **PO line** data consists of two object sets, **Coastl i ne** and **I nternati onal Boundary**:



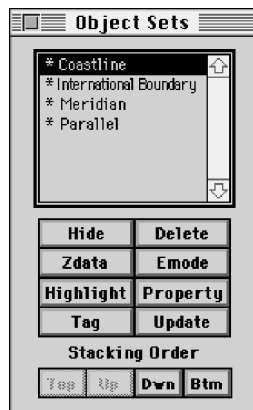
Press the **Close** button. Your figure now displays coastlines and international boundaries:



Close the **worldlo** dialog box. The worl d l o variables associated with the dialog box have been removed, but the plotted objects remain on the map.

Editing Object Properties

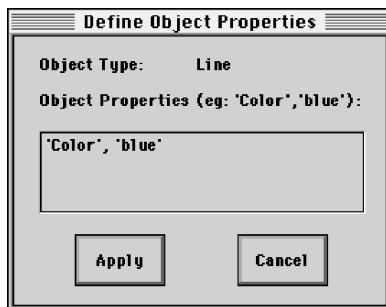
To view and edit properties of currently mapped objects, use the `objects` GUI. Select **Tools** and **Objects** from the menu bar. The **Objects Sets** dialog box appears, which lists all the objects on the current map axes. An asterisk next to an object indicates that it is visible, and an 'h' next to an object indicates that it is currently hidden.



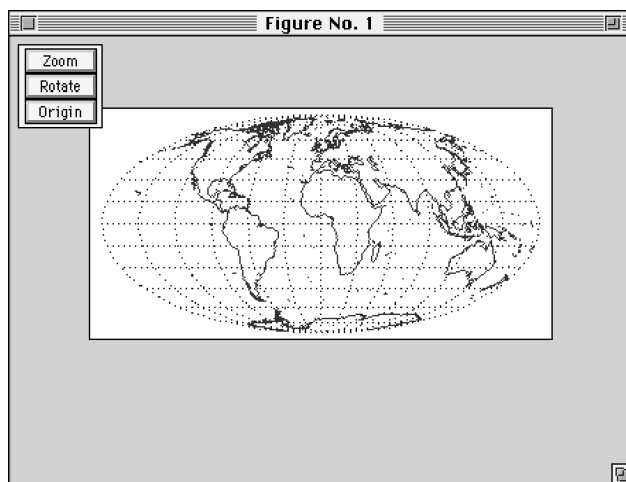
Select `International Boundary` and press the **Delete** button. A **Confirm Deletion** dialog box appears. Deleting an object set removes it from the current map axes. Press **Yes** to continue the deletion process.

The figure now displays only coastlines. Change the coastlines from black to blue by selecting `Coastline` from the list box and pressing the **Property** button. The **Define Object Properties** box appears. Multiple pairs of property names and values can be entered in this dialog box, but you would simply like to change the color of your coastlines from black to blue.

Type 'Col or' , ' bl ue' in the **Object Properties** edit box, and press **Apply**:



The figure now appears as follows:



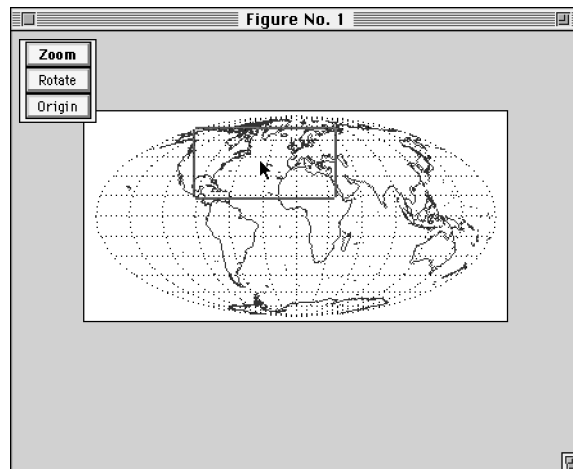
Close the **Objects Sets** dialog box.

Zooming in on a Map Display

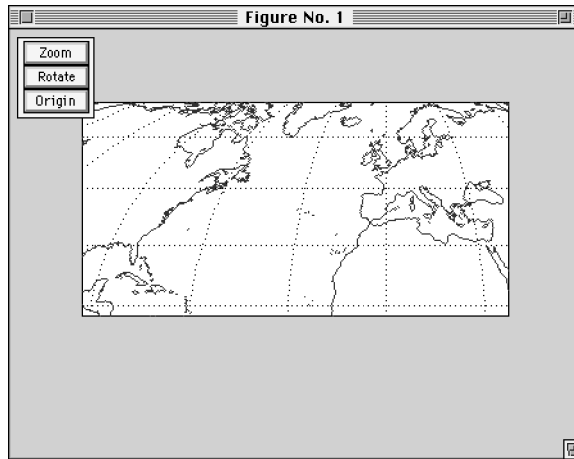
To get a closer view of the region you will be travelling in, you can use the zoom feature of maptool. Press the **Zoom** button in the upper left corner of the figure.

Zoom mode can also be activated by selecting **Tools⇒Zoom Tool** from the menu bar or with the command `panzoom`. A red zoom box will appear on the map once the zoom feature has been activated.

Drag the box until it encompasses the eastern half of the United States and most of Europe, as shown below. You can resize the box by clicking near a corner within the box and dragging the corner. Double-click inside the box to zoom in.



Here is the zoomed-in region. Click the **Zoom** button to end Zoom mode, or press **Esc** if you used the `panzoom` command.



You can zoom out again by selecting **Tools**⇒**Full View** from the menu bar.

For more information on the Zoom feature, see the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

Entering Workspace Commands

Now plot the cities that are on your flight schedule: Washington, D.C., London, Barcelona, Venice, and Hamburg. You will need two variables containing the latitude and longitude coordinates of these cities. Select **Session**⇒**Command** from the menu bar.

The **Workspace Commands** dialog box appears. This dialog box allows command line statements to be executed directly from `maptool`, rather than from the MATLAB command window. Enter the following:

```
citylats = [39.17; 51.59; 41.53; 45.27; 53.55];  
citylons = [-76.86; 0.04; 2.25; 12.21; 9.70];
```

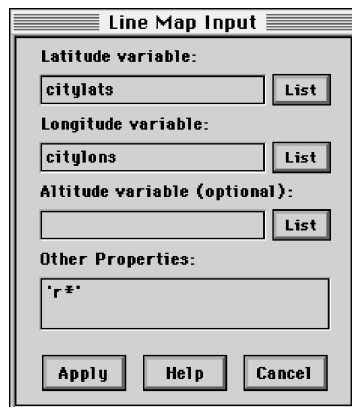

Press **Apply** when you have finished.



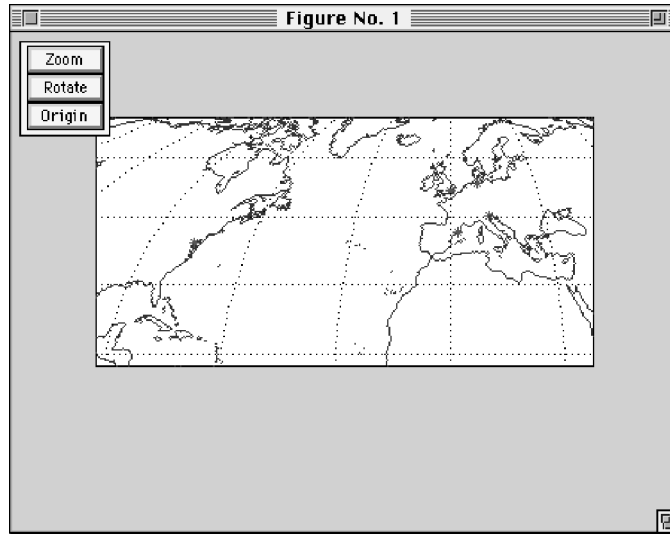
A **Status Report** box appears to indicate that the operations were successful. Press **OK**. Select **Session**⇒**Variables** from the menu bar, and you will see the two new variables in the workspace. Press **Close** to close the **Current Variables** box.

Plotting Line Objects

To plot the cities, choose **Map**⇒**Lines** from the menu bar. The **Line Map Input Box** appears. Enter the latitude and longitude variables, `citylats` and `citylons`. In the **Other Properties** edit box, type `'r*'` to plot a red marker at each city location. Press **Apply**.



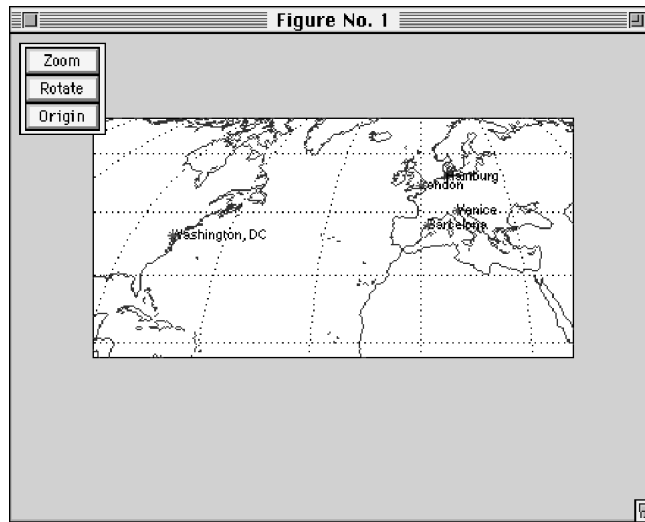
The city markers now appear on the map:



Plotting Text Objects

To plot city names, select **Map, Text** from the menu bar. The **Text Map Input** dialog box appears. In the **Text variable/string** edit box type `strvcat('Washington, DC', 'London', 'Barcelona', 'Venice', 'Hamburg')`. Enter `citylats` and `citylons` as the latitude and longitude variables. In the **Other Properties** edit box, type 'FontSize', 9, and press **Apply**. The city labels will appear on the map.

Here is the result:



For more information on using GUIs to plot text objects, see the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

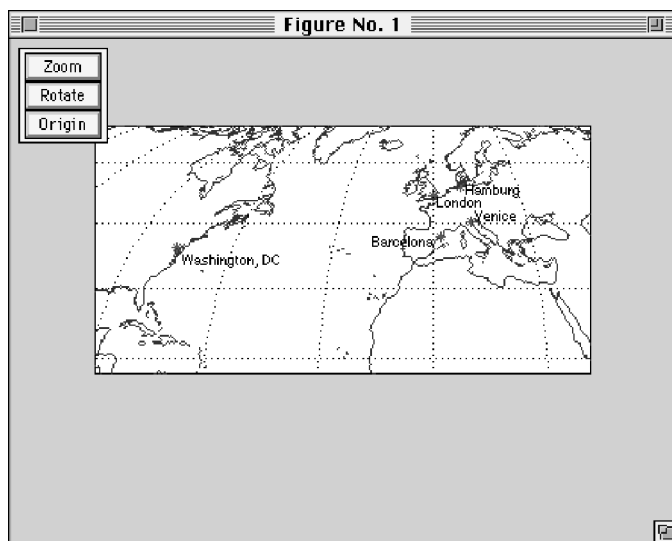
Editing Text Position

The properties of many mapped objects can be edited with the **Click-and-Drag Property Editor** tool. Pressing the **Alt** key and clicking on the object activates this functionality and allows for property manipulation through simple mouse clicks and drags. To move a city label away from its marker a bit, press the **alt** key and click on the text.

A **Click-and-Drag Property Editor** box for text objects appears:



Click **Drag**, and the label will now follow the mouse pointer as it is moved across the map display. Position the label where you like, and click to place the text. All the labels can be repositioned using the **Click-and-Drag Property Editor**:



For more information on using the **Click-and-Drag Property Editor** tool, see the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

Plotting Tracks and Calculating Surface Distances

To map out your flight plans, you can use the `trackui` tool. Select **Display⇒Tracks** from the menu bar. The **Define Tracks** dialog box appears.

Select **Great Circle⇒2 Point** mode. To specify Washington, D.C. as the starting point for your first track, press the **Mouse Select** button.

A crosshair appears on the map display. Position it over the city marker and click. The coordinates appear in the **Starting Point Lat** and **Lon** boxes. Using the **Ending Point Mouse Select** button, select London as your ending point.

In the **Other Properties** edit box, type ' Col or' , ' bl ack' , ' Li neWi dth' , 2:

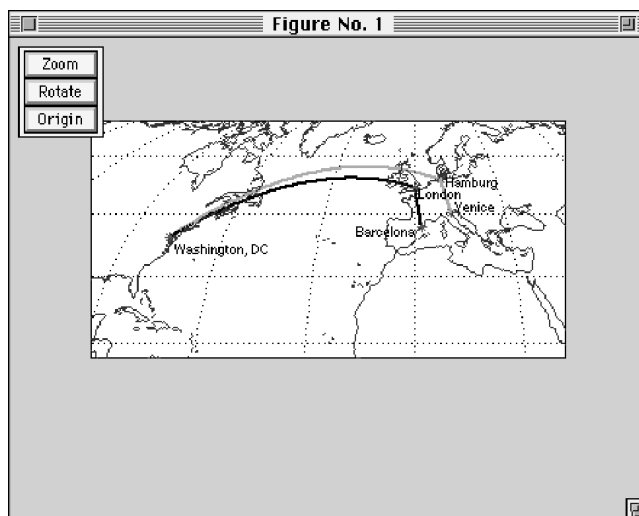


Press **Apply**, and the first flight track is displayed.

Now plot the flight from London to Barcelona by again using the mouse to enter those cities as your starting and ending points. Press **Apply**, and the second flight track is displayed. Now plot your flights back home. Use the mouse to specify a track from Venice to Hamburg.

To display your return flight in a different color, change the color to ' green' in the **Other Properties** edit box, then press **Apply**.

Finally, plot the flight from Hamburg to Washington, D.C. The map now displays your flight routes:



Press **Close** to close the **Define Tracks** dialog box.

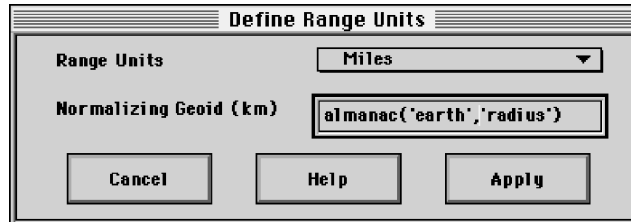
Now you are wondering which trip will cover more distance, your flights from Washington, D.C., to London to Barcelona, or your return flights from Venice to Hamburg to Washington, D.C. The answer can be easily determined using the `surfdist` GUI tool.

Select **Display**⇒**Surface Distances** from the menu bar. The **Surface Distances** dialog box appears. Select **Great Circle**⇒**2 Point** mode.

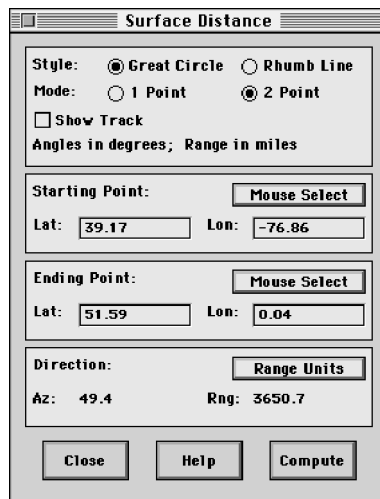
The tracks are already plotted, so leave the **Show Track** box unchecked. If the **Show Track** box were checked, the tracks would be displayed as the distances are calculated, but the tracks would be deleted when the **Surface Distances** tool is closed.

To calculate the distance of your first trip, you need to add the distance from Washington, D.C., to London and the distance from London to Barcelona. First, enter starting point (39.17,-76.86) and ending point (51.59,0.04) in the appropriate edit boxes. You want your answer in miles, so you need to change the range units.

Press the **Range Units** button, and the **Define Range Units** dialog box appears:



Select **Miles** from the pull-down menu, and press **Apply**. Press **Compute** to calculate the surface distance from Washington, D.C., to London. The answer appears next to the **Rng** label. The distance is 3,650.7 miles from D.C. to London:



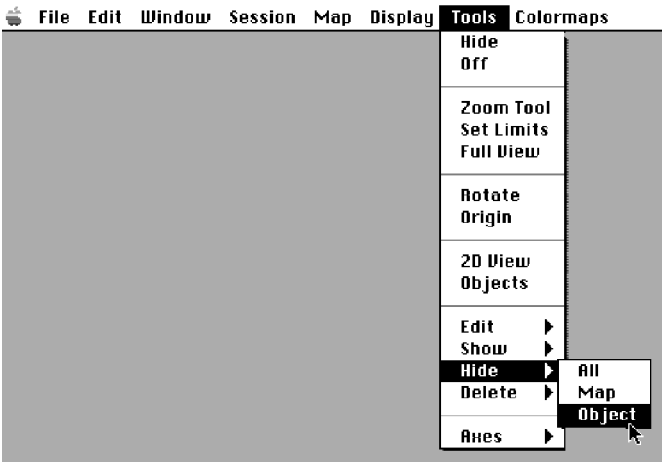
Next, enter starting point (51.59,0.04) and ending point (41.53,2.25) and press **Compute** to calculate the distance from London to Barcelona. The answer is 702.9 miles. The total distance covered on your trip from Washington, D.C., to Barcelona is $3,650.7 + 702.9 = 4,353.6$ miles.

Similarly computing the distances from Venice to Hamburg and from Hamburg to Washington, D.C., you find your trip back home covers $583.0 + 3,980.5 = 4,563.5$ miles.

Press **Close** to close the **Surface Distance** dialog box.

Hiding Mapped Objects

Suppose you now want to modify the map to detail the course of the cruise ship. First you want to hide the track lines, city markers, and city labels and leave only the coastlines displayed. This can be accomplished using the `objects` GUI, as was done previously to delete the international boundaries. It can also be done by selecting the **Tools⇒Hide⇒Object** option from the `maptool` menu bar:



The **Select Object** dialog box is activated, listing all currently mapped objects.



Select **G**reat **C**ircle **T**rack and press **O**k. The great circle tracks are then hidden. Do the same for the **t**ext and **l**ine objects, then close the **Select Object** box.

For more information on using GUIs to edit properties of mapped objects, see the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

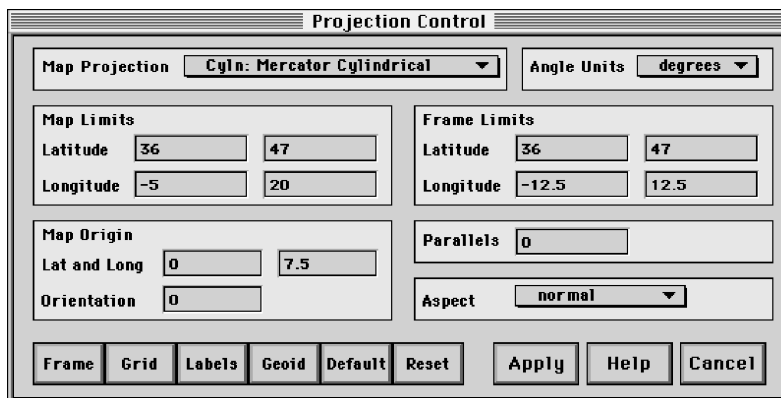
Editing Map Projection and Display Properties

You can display the navigational tracks of the ship by identifying waypoints and plotting them on a Mercator projection, then connecting the points by using the track command. This will result in a series of rhumb line (constant direction) tracks. The tracks appear as straight lines on the Mercator projection, but they appear curved on others.

Select **Display**⇒**Projection** from the menu bar, and change the map projection to a Mercator Cylindrical. Change the map latitude limits to [36 47], the longitude limits to [-5 20], and the map origin to [0 7.5 0]. Always reset the appropriate map properties when switching projections or when changing map or frame limits. In this case, the frame limits should be reset for the new map limits.

Clear the frame limits boxes, and press the **Default** button to recalculate the frame limits for the current entries.

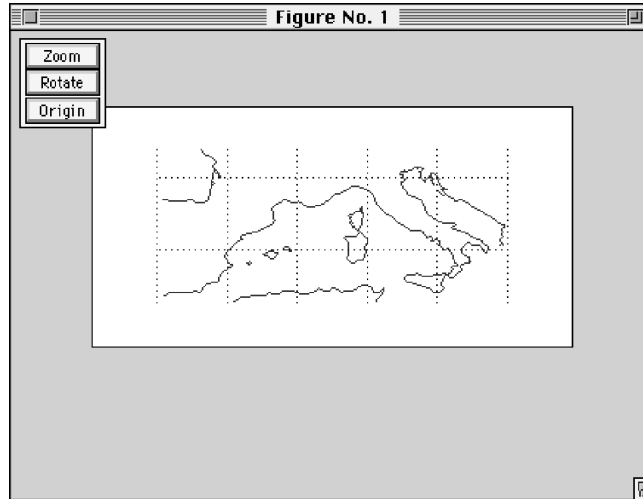
The **Projection Control** dialog box should now appear as follows:



Press the **Grid** button to bring up the **Map Grid Properties** dialog box. Press **Meridian and Parallel Settings**, and change the longitude and latitude locations of the grid lines to 5.

Press **Accept** to return to the **Map Grid Properties**, and then press **Accept** to return to the **Projection Control** box.

Press **Apply** to re-project the map with the new settings. Note how the parts of the map outside the limits you specified are now automatically trimmed:



For more information on using the **Projection Control** box to edit map projection and display properties, see the descriptions for `axesm`, `axesmui` in the “GUI Reference” section of the *Mapping Toolbox Reference Guide*.

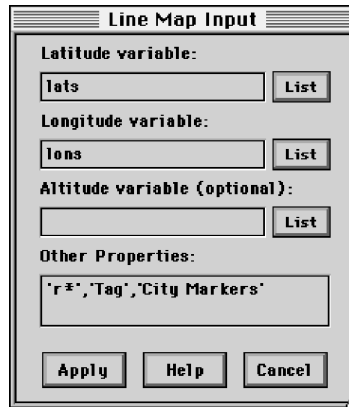
To plot city markers and labels as you did for the previous map, first select **Session**⇒**Command** from the menu bar. Enter the following in the **Workspace Command** dialog box:

```
lats = [41.53 43.44 43.33 42.19 41.01 45.27];
lons = [2.25 7.25 11 12.47 14.22 12.21];
```

Press **Apply** when done.



Next, select **Map⇒Lines** from the menu bar. Enter the **Latitude variable** `lats` and the **Longitude variable** `lons` in the appropriate edit boxes, along with the line style `'r*'` and `'Tag', 'City Markers'` in the **Other Properties** edit box. Press **Apply** when done.



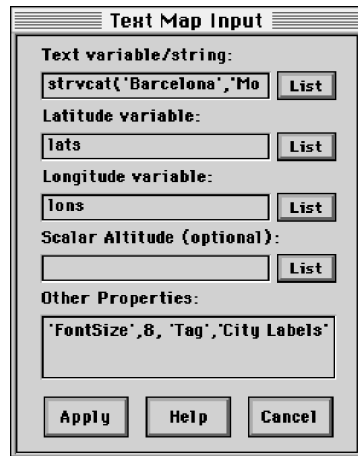
Finally, select **Map, Text** from the menu bar. Enter the following in the **Text variable/string** edit box:

```
strvcat(' Barcelona', ' Monte Carlo', ' Livorno', ' Rome', ' Naples', ...
        ' Veni ce')
```

Enter the variables `lats` and `lons` in the **Latitude variable** and **Longitude variable** edit boxes, respectively. In the **Other Properties** edit box enter the following:

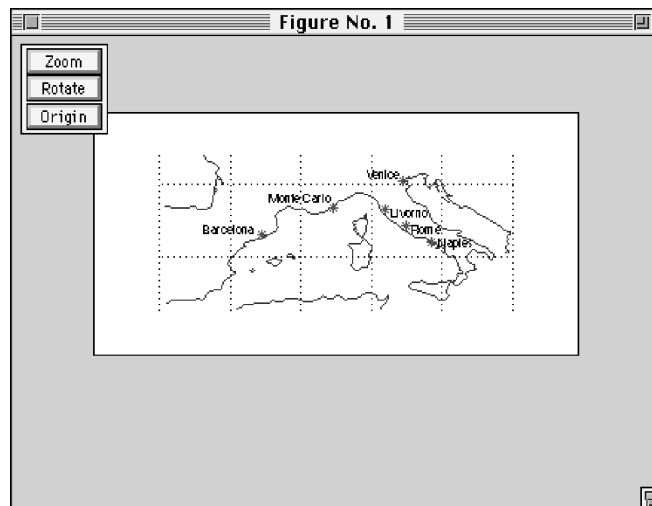
```
' FontSi ze', 8, ' Tag', ' City Labels'
```

Press **Apply** when you have finished.



City markers and labels now appear on the map. The **Click-and-Drag Property Editor** can be used to easily edit the label locations to be more pleasing to the eye.

Your map might then look something like this:



Selecting Track Waypoints

To select track waypoints for the cruise ship, use the `inputm` command. Select **Session⇒Command** from the menu bar, and enter the following in the **Workspace Command** box:

```
[waylat, waylon] = inputm;
```

Press **Apply** when done.

A cross hair appears on the map display, allowing you to use the mouse to click on the desired waypoints. You know that you will be sailing from Barcelona, around the islands of Mallorca and Menorca, to Monte Carlo, then to Livorno, Rome, Naples, through the Straits of Messina, and up the Adriatic to Venice.

Use the mouse to click on points that trace out such a path, then press **Return**. The **Status Report** box appears to let you know the operation was successful. Press **OK**.

The variables `waylat` and `waylon` are now in the workspace. This can be confirmed by selecting **Session⇒Variables** from the menu bar to view the current workspace variables.

Creating a Navigational Track

Now you can use the `track` command to connect your waypoints. Select **Session⇒Command** from the menu bar, and enter the following in the **Workspace Command** box

```
[tracklat, tracklon] = track([waylat waylon]);
```

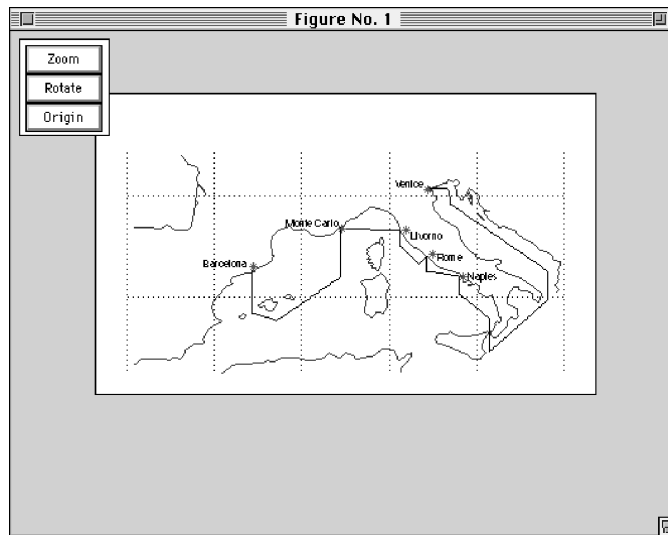
Press **Apply** when done. Again the Status Report box appears. Press **OK**. The variables `tracklat` and `tracklon` are now in the workspace.

For more information on connecting waypoints with track segments, see the “Mapping Reference” section of the *Mapping Toolbox Reference Guide*.

Displaying a Navigational Track

To plot the tracks, select **Map⇒Lines** from the menu bar. Enter track1 at in the **Latitude variable** edit box and track1 on in the **Longitude variable** edit box. In **Other Properties**, enter 'Tag', ' Navigational Track', ' Col or', ' black', and press **Apply**.

The track appears on your map display:



Plotting Patches

To make your map more colorful, you can plot the country patches from the worldl o workspace using the mlayers GUI. Select **Display⇒Projection** from the menu bar, use the **Grid** button to turn the grid off, then press **Apply**.

To plot the patches, select **Session⇒Layers⇒World LoRes** from the menu bar. When the **worldl o** dialog box appears, select P0patch, and press **Plot**. After the country patches have been plotted, close the **worldl o** dialog box.

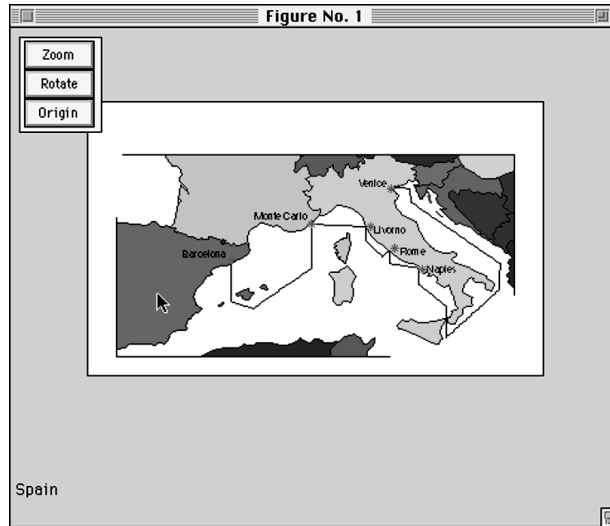
Changing the Stacking Order of Mapped Objects

The country patches were plotted in the same z plane as the city markers and labels. You can place the markers and labels on top of the patches by changing the stacking order. Select **Tools⇒Objects** from the menu bar.

The **Object Sets** dialog box appears:



Select **City Markers**, and press **Top**. Then select **City Labels**, and press **Top**. Close the **Object Sets** box. The markers and labels are now on top of the patches:



Your map now displays the cities you will be visiting and the track the ship will take. Clicking on a country displays that country's tag in the lower left corner of the figure window.

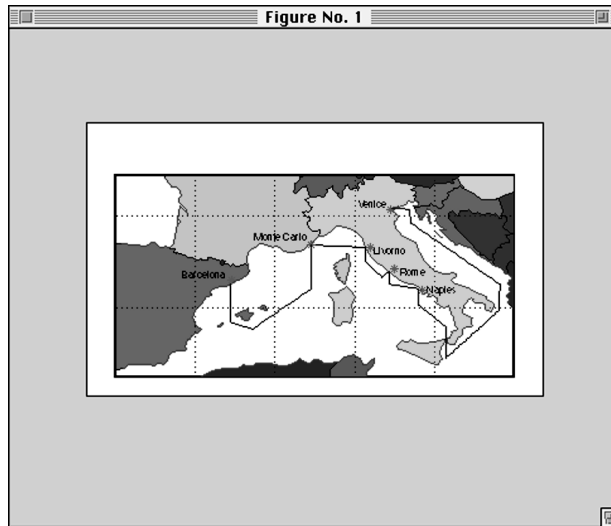
Hiding the Mouse Buttons

Select **Tools, Hide** from the menu bar to hide the **Zoom**, **Rotate**, and **Origin** buttons on the figure window.

Toggling the Map Grid and Frame On

To complete your map, you want to display the grid and frame. This can be done through the **Projection Control** dialog box, or by simply selecting **Display, Grid** and then **Display, Frame** from the menu bar.

The final map looks like this:



If you are satisfied with your map, you may want to select **File⇒Save** from the menu bar to save the figure window in an M-file. When you execute the M-file, the figure and uicontrols will be recreated to their original appearance.

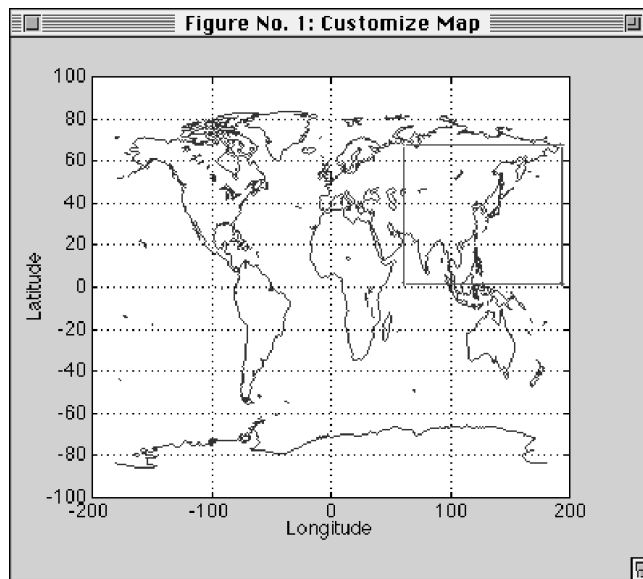
Constructing Personalized Map Data with GUIs

Trimming an Existing Data Set

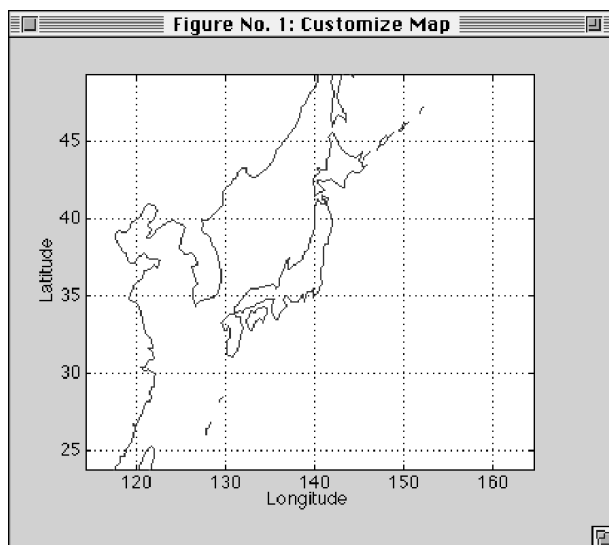
You can use the `maptrim` tool to trim a map dataset. For example, you want to work with just a portion of the coastline data found in the `coast` workspace, say the region around Japan. Use the `maptrim` tool to trim the coastline data to this region, then save the trimmed dataset as a new vector or matrix map.

```
load coast  
maptrim(lat, long)
```

A new figure window displays the unprojected coastline data. Drag the zoom box to the region around Japan.



Double-click in the box to zoom in. Repeat this process until the area of interest fills the figure window.



Select **Customize⇒Limits** to specify the exact limits of the region of interest. A dialog box appears with the latitude and longitude limits of the current display. Set the latitude limits to [24 48] and the longitude limits to [123 149]. Press **OK**.

To save this trimmed region as vector data, choose **Customize⇒Save As⇒Line** or **Customize⇒Save As⇒Patch** from the menu bar. A dialog box will appear prompting you to enter variable names for your new dataset.

The `maptrim` tool can also be used to convert data from vector format to matrix format. To convert the trimmed region around Japan, choose **Customize**⇒**Save As**⇒**Regular Surface** from the menu bar. Save the trimmed map data under the variable name `map`, and the map legend under variable name `maplegend`. Enter a scale of 5 cells per degree. Press **OK**.

The screenshot shows a standard MATLAB-style dialog box titled "Enter the Surface Map variable names". It contains the following fields and controls:

- Map Variable:** A text input field containing the string "map".
- Map Legend Variable:** A text input field containing the string "maplegend".
- Scale (cells/degree):** A text input field containing the number "5".
- Latitude Limits (Optional):** An empty text input field.
- Longitude Limits (Optional):** An empty text input field.
- Buttons:** "Cancel" and "OK" buttons at the bottom.

You now have a regular matrix map of Japan in your workspace. The **Customize Map** figure window can now be closed.

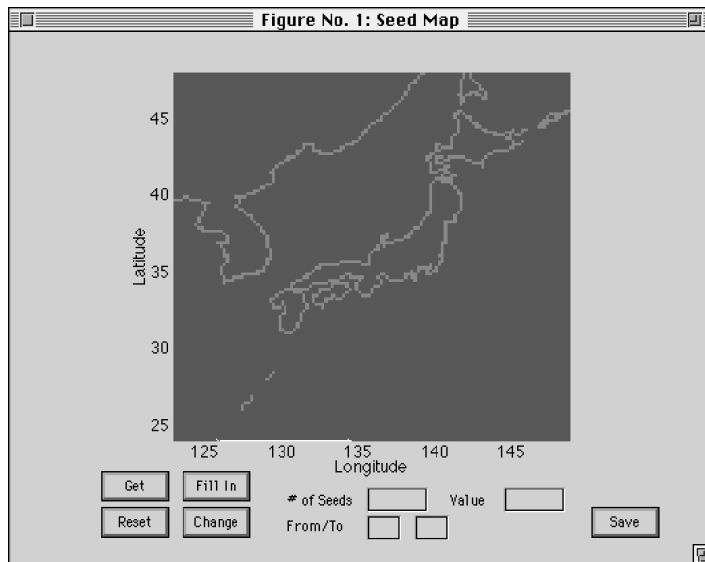
Encoding a Regular Surface Map

Encoding is the process of filling in specific values in regions of a matrix map up to specified boundaries. For example, you can fill in the interior of an island, bounded by its coastline, with the value six. Encoding entire regions at one time allows indexed maps to be created quickly.

A regular matrix map can be encoded using the `seedm` tool. To encode the map of Japan just created, activate the `seedm` tool by typing the following:

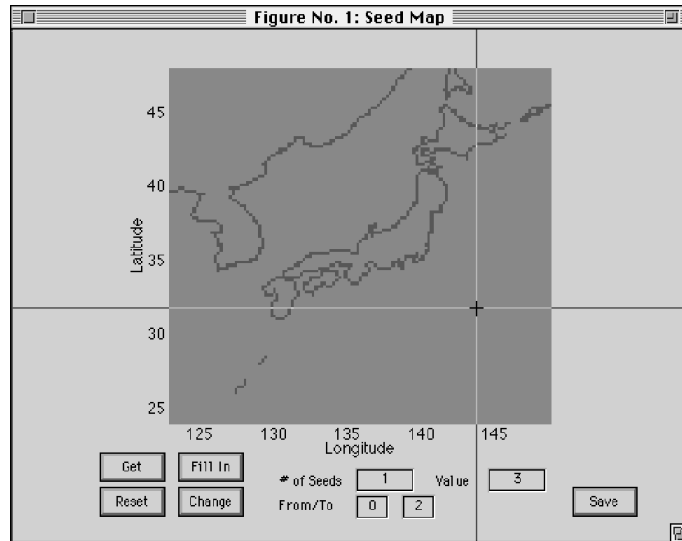
```
seedm(map, maplegend);
```

A new figure window displays the surface map of Japan.



Currently, your map consists of 1's where there is a coastline and 0's elsewhere. Suppose you would like different values for four types of data: coastlines = 1, Japan = 2, water = 3, and other countries = 4. Enter the values 0 and 2 in the **From/To** boxes and press **Change**. This will change all values of 0 to 2.

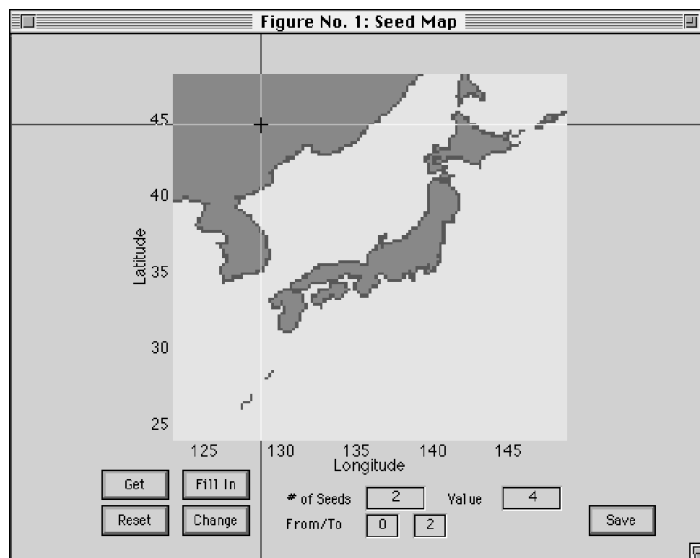
To encode the regions of water, enter a 1 in the **# of Seeds** box and a 3 in the **Value** box. Press **Get**, and a cross hair will appear on the map. Position the cross hair over a region of water and click.



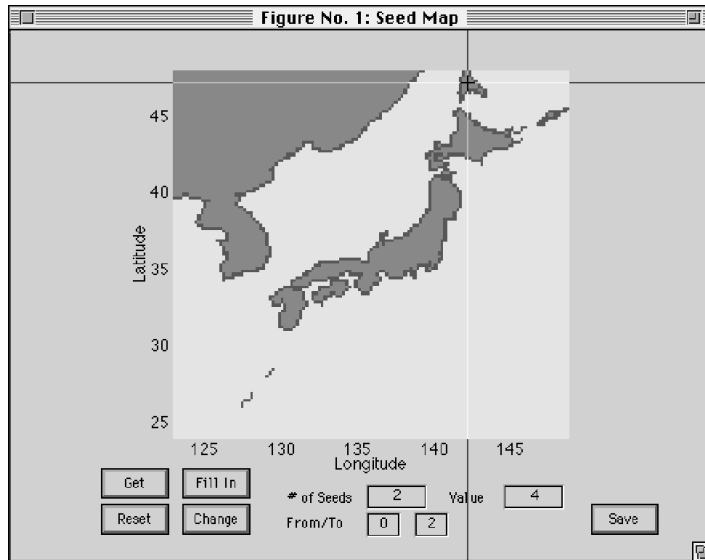
Press **Fill In**, and all values corresponding to water will be changed to 3.

Encode the other countries, there are three regions that will need to be filled with the value 4. Encode the large region of Russia, China, North Korea, and South Korea in the upper left of the map and the Russian Island of Sakhalin at the top of the map in a single step by using two seeds. Enter a 2 in the **# of Seeds** box, and a 4 in the **Value** box.

Press **Get** and the cross hair appears. Click in the first region at the upper left of the map.

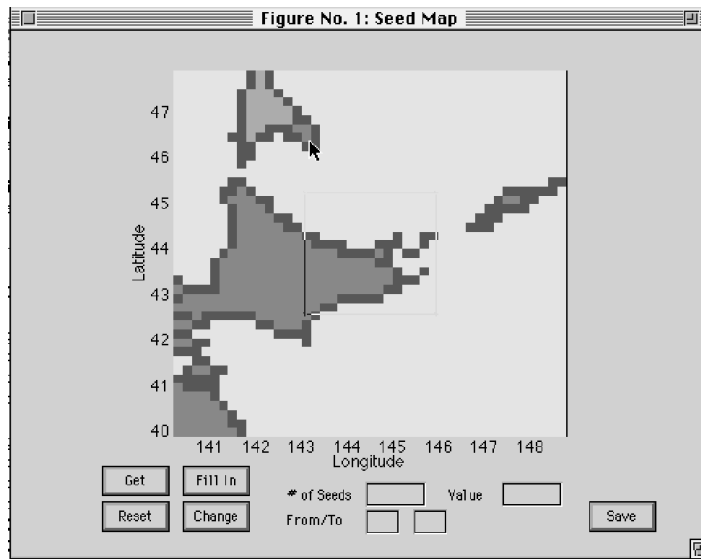


For the second seed, click inside Sakhalin Island.



Press **Fill In**, and the values corresponding to those two regions will be changed to 4. The last region you need to encode is the small Kuril island that appears to the east of Hokkaido in the upper right corner of the map.

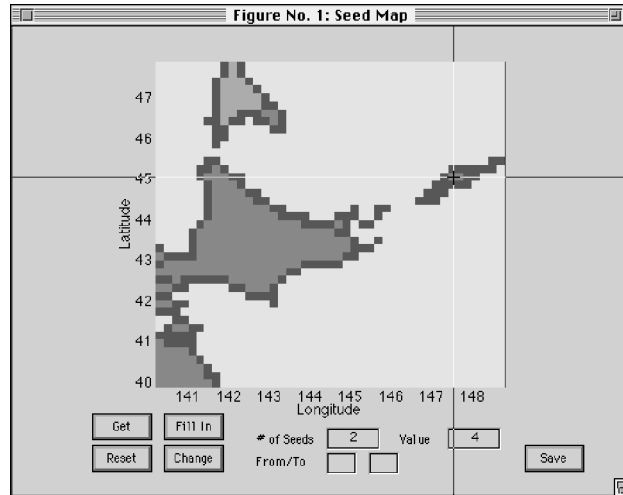
Drag the zoom box over the island and double-click to zoom in.



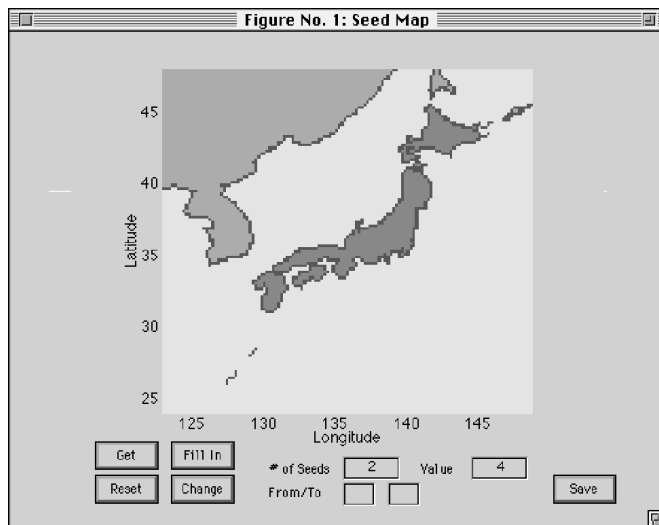
Because the island is so small, and much of its shape consists of coastline, this zoomed-in view will make it much easier to select the interior of the island, rather than its coast.

From this view, you can also see that you missed a spot on Sakhalin Island (see mouse pointer in the picture above), so you will need two seeds for this step.

Enter a 2 in the **# of Seeds** box and a 4 in the **Value** box. Press **Get** and click inside the missed region of Sakhalin Island and inside the Kuril island.



Press **Fill In** and the values corresponding to those two regions will be changed to 4. The final map appears as follows:



Press the **Save** button and enter the variable name **map** to replace the old matrix map of Japan with this new encoded map. The **Seed Map** dialog box can now be closed.

Creating a Personalized Colormap

Now you can use the `colorm` tool to make a custom colormap for your map of Japan. To activate the `colorm` GUI, type:

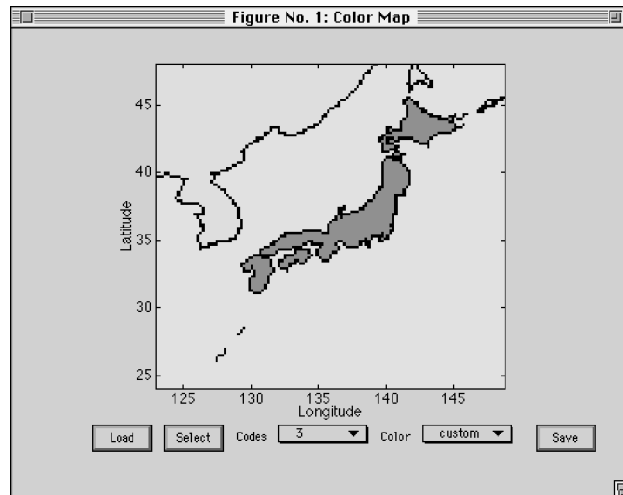
```
colorm(map, maplegend)
```

A new figure window displays the map. Recall that the values for this matrix map are as follows: coastlines = 1, Japan = 2, water = 3, and other countries = 4.

To change the coastlines to black, select 1 from the **Codes** menu, and choose black from the **Color** menu. Next, make Japan a light purple by selecting 2 from the **Codes** menu and choosing custom from the **Color** menu. The **Custom Color** GUI appears.

Select a custom color, and choose **OK**. Japan will change to the new color. Next, change the color of all other countries to a light beige by selecting 4 from the **Codes** menu, and custom from the **Color** menu. Use the **Custom Color** GUI to choose a light beige color, then press **OK**.

Finally, to change the color of water to a light blue, select 3 from the **Codes** menu and custom from the **Color** menu. Choose a light blue from the **Custom Color** GUI and press **OK**. The map of Japan now appears:



Press the **Save** button, then press **OK** to accept the default name `cl rmap` for the new colormap variable. The **Color Map** dialog box can now be closed.

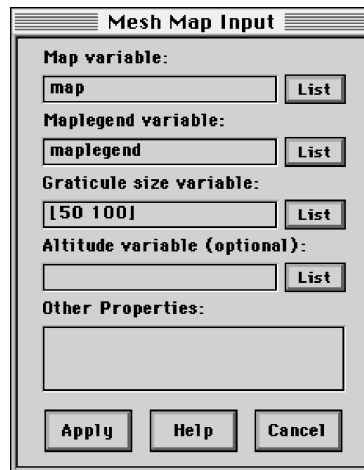
To display the map of Japan, type `axesm`.

In the **Projection Control** box, choose a Miller Cylindrical projection and define the map latitude and longitude limits to be `[24 48]` and `[123 149]`, respectively.

Use the **Grid** button to turn the grid on, then press the **Parallel and Meridian Settings** button and set the longitude locations and latitude locations to 5 to display grid lines every 5 degrees.

Use the **Label** button to turn the meridian and parallel labels on. Change the label format to none. Press **Parallel and Meridian Settings** button and change the longitude and latitude label locations to 5 to display labels every five degrees. Press **Apply** to apply the settings to the map axes.

To plot the map of Japan using the **Mesh Map Input** dialog box, type `meshm`.

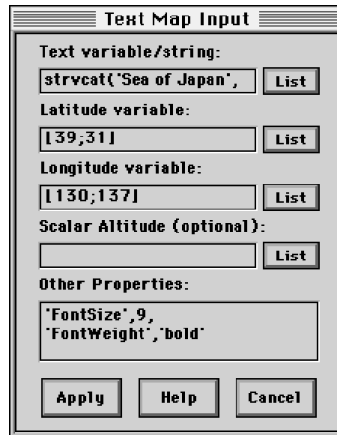


Your variable names are the same as the defaults, so press **Apply** to plot the map. After the map is plotted, type `colormap(cl rmap)`. This will set the current colormap to the customized one you created. To annotate the major bodies of water in the display, type `textm`.

The **Text Map Input** dialog box appears. In the **Text variable/string** edit box, enter the following:

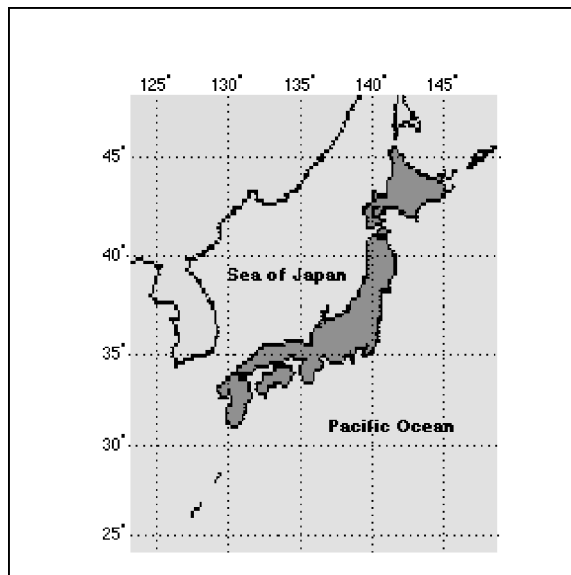
```
strvcat(' Sea of Japan' , ' Paci fic Ocean' )
```

In the **Latitude variable** edit box enter [39; 31], and in the **Longitude variable** edit box enter [130; 137]. In the **Other Properties** edit box, enter ' FontSi ze' , 9 and ' FontWei ght' , ' bol d' .



Press **Apply** to plot the text.

The final display of your customized map of Japan appears as follows:



External Data Interface

Working With External Data	7-2
Global Vector Data	7-3
The Digital Chart of the World	7-3
U.S. Vector Data	7-14
TIGER/Line	7-14
TIGER Thinned Boundary	7-18
Global Gridded Elevation Data	7-23
ETOPO5 and TerrainBase	7-23
Digital Chart of the World DEMs	7-28
U.S. Gridded Elevation Data	7-32
Astronomical Data	7-36

Working With External Data

While the Mapping Toolbox atlas data described in Chapter 3 is suitable for world or regional maps, more detailed data is usually required for smaller scale maps. The best sources of such data are often national mapping agencies, who make their data available to the public at little or no charge, often over the Internet or on CD-ROM. United States government publications are generally free of copyright, allowing free re-use of the data.

Although the Mapping Toolbox can easily display such geographic data once it is in the MATLAB environment, importing it is often difficult because of complicated data file formats. To make these detailed datasets readily accessible, The Mapping Toolbox provides a number of external data interface functions. These functions allow you to read the data into Mapping Toolbox data types such as regular matrix maps and geographic data structures.

A wide variety of data is accessible through these external interface functions. There are vector map data of political entities ranging from national and local government boundaries, infrastructure such as roads, railroads, airports and utilities, and geographic data such as coastlines, rivers, elevation contours, and land use. The Earth's surface and ocean elevations are available on regular grids at a number of different resolutions. Astronomical data in the form of star databases is also available.

Many other types of data can be imported to MATLAB beyond the specialized data file formats accessed by the external interface functions. The most important are mapped scientific data in the form of HDF files and JPEG files. See the MATLAB documentation for the `imfinfo` and `imread` functions for more information on how to read these file formats.

Global Vector Data

The Digital Chart of the World

The most detailed, publicly available set of global vector data with consistent coverage of important map features is the Digital Chart of the World (DCW). It contains vector data scanned from printed Operational Navigation Charts (ONC) and Jet Navigation Charts (JNC). The ONC charts were compiled at a scale of 1:1,000,000 and the JNC at 1:2,000,000. The publisher of the DCW was the U.S. Defense Mapping Agency, now incorporated into the U.S. National Imagery and Mapping Agency.

The DCW contains data for the entire world in separate layers of coverage, including political and ocean features, drainage, hypsography, land cover and vegetation, populated places, roads, railroads, airports, transportation structure, and cultural landmarks. The features are formatted as patches, lines, points, or text.

The DCW was published in 1992 on four CD-ROMs. A new version, called VMAP0, had been scheduled for release in 1995, but was not available as of early 1997. For more information on purchasing the DCW, see the *dcwdata* entry in the “External Data Reference” section of the *Mapping Toolbox Reference Guide*.

The Mapping Toolbox provides high-level functions that compile and extract information from the DCW into Mapping Toolbox geographic data structures. The following sections introduce these functions and show how to work effectively with the DCW data.

Looking Up Names and Locations in the DCW Gazetteer

One source of information in the Digital Chart of the World is the Gazetteer. This is an extensive collection of place names and types within a library. The NOAMER library, containing the data for North America, has 37696 names in its gazette. The entire DCW gazette has more than 100,000 names. It can be used to quickly find the locations of places and can help to pick the latitude and longitude limits for other DCW functions.

The Mapping Toolbox interface to the DCW gazette is the `dcwgaz` function. Assuming that the EURNASIA CD is available and mounted on the desktop of a Macintosh computer, you can search for the geographic location of the city of Apatin in Yugoslavia by typing:

```
dcwgaz(' EURNASIA' , ' apati n' )
APATIN
ans =
           type: 'text'
  otherproperty: {1x2 cell}
           tag: 'Built up area'
         string: 'APATIN'
        altitude: []
             lat: 45.6660
             long: 18.9830
```

On other computers, you must provide a device name as the first argument. The string you provide will depend on how the CD-ROM is mounted. For a Windows PC, the command might be

```
dcwgaz(' F: ' , ' EURNASIA' , ' apati n' )
```

and on a Unix computer with the CD-ROM mounted as `/cdrom`, you should type:

```
dcwgaz(' /cdrom/' , ' EURNASIA' , ' apati n' )
```

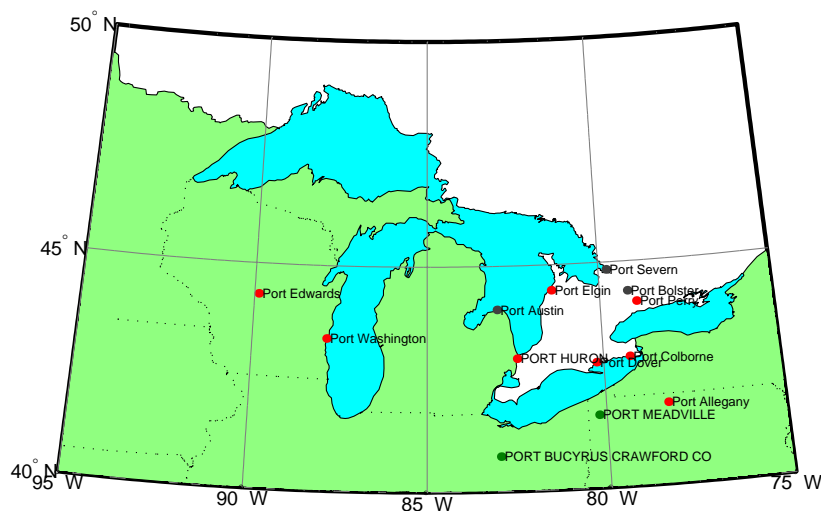
See the documentation that came with your CD-ROM drive for more information on how disks are mounted.

Matches are displayed on screen during the search. The output is returned in a Mapping Toolbox geographic data structure, with the tag identifying the database layer name within the DCW. If there is more than one entry matching the name, the results will be contained in an arrayed structure.

For example, the following command prints to the screen the 95 names in the North American library that begin with the word ' port ', and returns the structures `txtstruc` and `ptstruc` that can be displayed using `display` or `mlayers`.

```
[txtstruc, ptstruc] = dcwgaz(devicename, ' NOAMER' , ' port ' );
```

Here is a map that shows the “ports” around the Great Lakes:



Importing DCW Themes

Although the gazette feature of the DCW is extensive, the DCW is designed primarily as a database of geographic information for more general mapping applications. The `dcwdata` function is used to access the map data.

The volume of data on the DCW is so large that you must limit your requests to a reasonable amount. First, select the geographical area of interest by giving the name of one of the libraries in the DCW. There is one CD for each library: 'NOAMER' (North America), 'EURNASIA' (Europe and Northern Asia), 'SOAMAFR' (South America and Africa), and 'SASAU' (Southern Asia and Australia).

You may also need to specify the device name of the CD, just as you did for the `dcwgaz` function. The geographic region of interest is further restricted, providing latitude and longitude limits in units of degrees.

You also need to specify which “theme,” or type of data, you wish to extract. Themes are layers of data related to a kind of feature, like roads and drainage, and are listed by two-letter strings. If you enter a code that isn't recognized, a list of valid codes and their descriptions will be displayed.

Finally, you need to indicate what type of graphic objects you want (patches, lines, points, or text). These are called topology levels.

If the 'NOAMER' CD is available, you can see the list of available themes by entering an incorrect one, like '?'. The following example is for a Macintosh computer, so no device name is needed. The latitude and longitude limits correspond to the Cape Cod region:

```
Struc = dcwdata('NOAMER', [41 44], [-72 -69], '?', 'line')
??? Error using ==> dcwdata
Theme not present in library NOAMER
```

Valid two-letter theme identifiers are:

```
PO: Political/Oceans
PP: Populated Places
LC: Land Cover
VG: Vegetation
RD: Roads
RR: Railroads
UT: Utilities
AE: Aeronautical
DQ: Data Quality
DN: Drainage
DS: Supplemental Drainage
HY: Hypsography
HS: Supplemental Hypsography
CL: Cultural Landmarks
OF: Ocean Features
PH: Physiography
TS: Transportation Structure
```

Assuming that you are using a Windows PC, with the 'NOAMER' CD-ROM in the 'F:' drive, you can read the Political/Ocean line features for the same Cape Cod region by entering the following:

```
POLine = dcwdata('F:', 'NOAMER', [41 44], [-72 -69], ...
                'P0', 'line')
POLine =
    1x12 struct array with fields:
        type
        otherproperty
        altitude
        lat
        long
        tag
```

The result is a Mapping Toolbox geographic data structure. Look at the geographic limits by concatenating the data with `extractm`.

```
[lat, long] = extractm(POLine);
[max(lat) max(long); min(lat) min(long)]
ans =
    45.0003   -65.0000
    40.0000   -75.0000
```

Notice that this region is larger than was requested. The DCW tiles the world into 5-by-5 degree quadrangles. When a tile overlaps the requested region, `dcwdata` extracts the data for the entire tile. Likewise, if you request a point location, the entire 5-by-5 quadrangle is returned. All objects are cut at the edges of the tiles.

The kind of information in an element of the structure is described by the tag field. There are 8 kinds of boundary lines in this region:

```
unique(strvcat(P0line.tag), 'rows')
ans =
; None
Administrative boundary, primary; Definite
Administrative boundary, primary; Poly closure line
Administrative boundary, primary; River, when boundary
Coastal Closure Line; None
Coastline; Definite
Connector; None
International boundary, Dejure; Definite
```

You can extract more than one topology level using a cell array of level names. Extracting patches and points is more time-consuming, so this will take longer than the previous example. The following example assumes that the 'NOAMER' CD-ROM is mounted on a Unix computer as '/cdrom':

```
[P0patch,P0point,P0text] = dcwdata('/cdrom','NOAMER',[41 44]...
                                   [-72 -69], 'P0',...
                                   {'patch' 'point' 'text'});

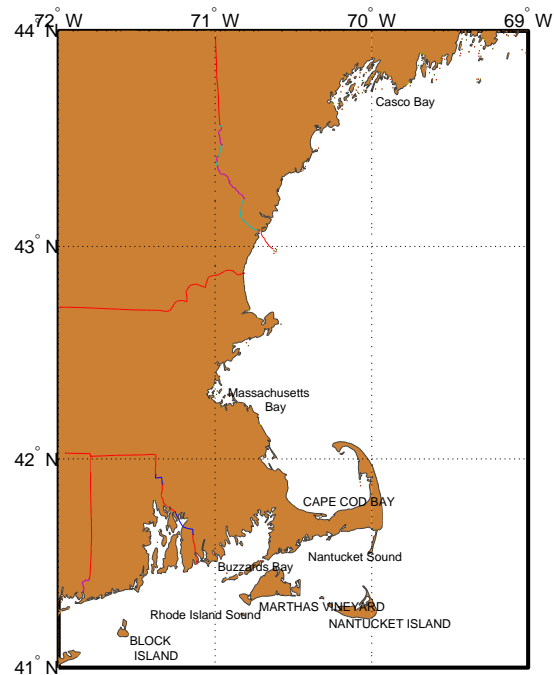
P0patch(1)
ans =
        type: 'patch'
otherproperty: {}
altitude: []
lat: [6x1 double]
long: [6x1 double]
tag: 'Land'
tag2: 'N; US; MAINE'
tag3: 'North America; United States; MAINE'
```

The patch and point data in this example have primary tags of 'Land' or 'Open Ocean'. There are also alternate tags containing high-level political units such as nations and subordinate units such as provinces or states. The alternate tags are available in both abbreviated and expanded forms.

The following commands plot the political and ocean features on a Mercator projection, hiding the ocean patches and the tile boundary lines. They also override the default colormap used for the land patches and remove the edge color to hide the tile join:

```
axesm('MapProjection','mercator',...
      'MapLatLimit',[41 44], 'MapLonLimit',[-72 -69],...
      'MLabelLocation',1, 'MLineLocation',1,...
      'PLabelLocation',1, 'PLineLocation',1);
frame; gridm; mlabel; plabel

hP0patch = displaym(P0patch); hP0line = displaym(P0line);
hP0point = displaym(P0point); hP0text = displaym(P0text);
hidem(handlem('Open ocean')); hidem(handlem('; None'))
brown = [.8 .5 .2];
set(handlem('Land'),'FaceColor',brown,'EdgeColor','none')
set(hP0text,'FontSize',6)
```



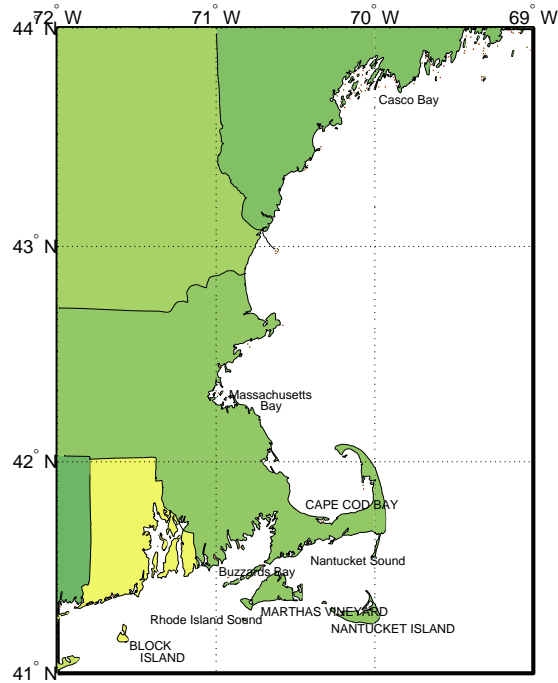
Notice how much more detailed this map is than even the best of the atlas data provided in the Mapping Toolbox. The highest resolution atlas data is in the `usahi` workspace, which is limited to the United States. The DCW covers the entire world at this level of detail.

The alternate tags can show the same data in a different display. You can make the states different colors if you copy the alternate tags containing the state names into the primary tag field. You may find it convenient to write a small function to trade the tags without overwriting:

```
function struc = switchfields(struc, fromfield, tofield)

for i=1:length(struc)
    eval(['temp = struc(i).' tofield ';' ]);
    eval(['struc(i).' tofield ' = struc(i).' fromfield ';' ]);
    eval(['struc(i).' fromfield ' = temp;']);
end % for
```

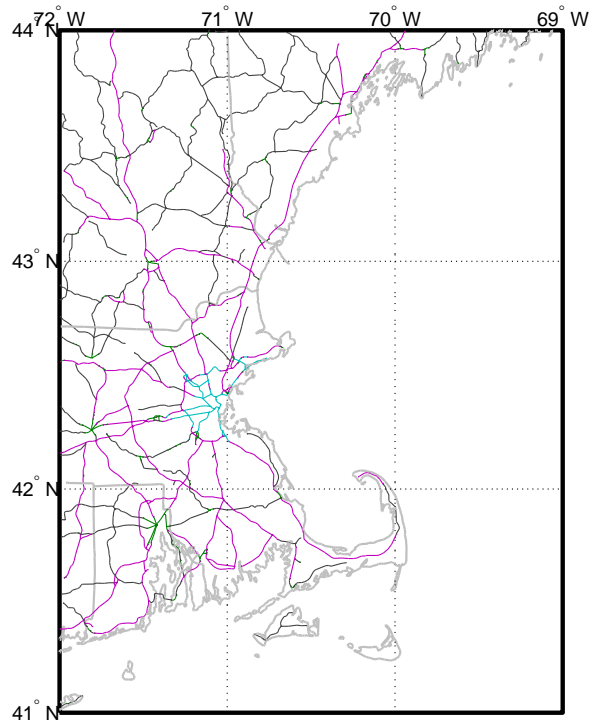
Here is a new map of the data plotted with the summer colormap:



The DCW does not contain all topology levels in each theme. Roads, for example, have no patches. If data is requested for topology levels that are not available, the function issues a warning and returns an empty matrix. Here is a display of the road line data for the Cape Cod map:

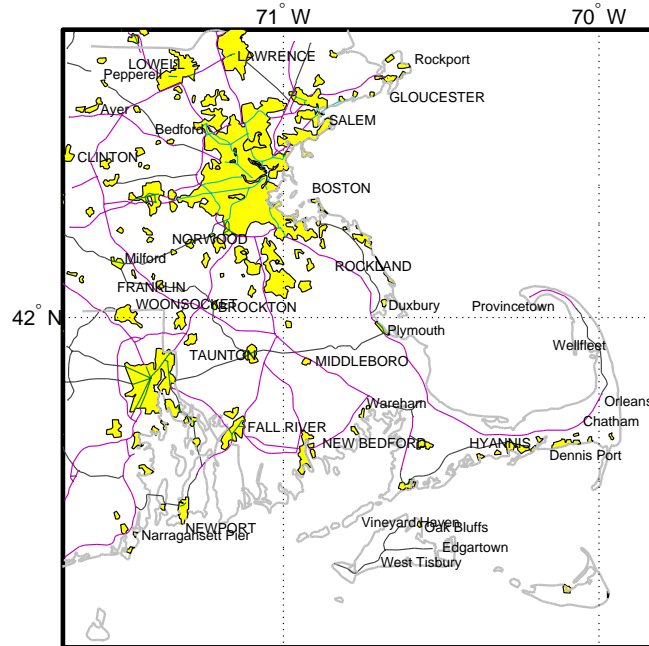
```
[RDpatch, RDline, RDpoint, RDtext] = dcwdata('/cdrom', 'NOAMER', ...
                                             [41 44], [-72 -69], ...
                                             'RD', {'all'});
```

```
clma
hPoline = displaym(Poline);
hidem(handlem(' ; None'))
set(hPoline, 'Color', [1 1 1]*0.75, 'LineWidth', 1)
hRDline = displaym(RDline);
```



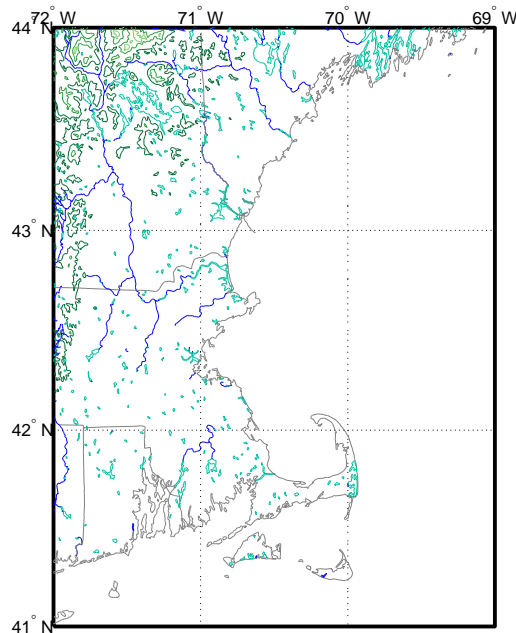
The roads patterns are clearly related to settlement patterns. The network of roads around Boston gives away its location. The DCW also includes a populated places theme, containing the boundaries of built-up areas. Extract the 'PP' theme, reduce the displayed area to Boston and Cape Cod, and plot the populated places:

```
[PPpatch, PPtext] = dcwdata('/cdrom', 'NOAMER', [41 44], ...
                             [-72 -69], 'PP', {'patch' 'text'});
setm(gca, 'MapLatLi mi t', [41.13 42.75], ...
      'MapLonLi mi t', [-71.7 -69.8])
hPPpatch = displ aym(PPpatch); set(hPPpatch, 'FaceCol or', 'y')
hPPtext = displ aym(PPtext); set(hPPtext, 'FontSi ze', 6)
```



There is more data describing man-made features, including airports, railroads, utilities, cultural landmarks, and transportation structure. The DCW also contains data for the natural world. The drainage and elevation contours are perhaps the most extensive data themes.

Here is a map of the drainage and contour data:



Reading DCW Files Directly

The high-level `dcwdata` function is the most convenient way to read data from the DCW because it integrates information from many different files in different directories. Individual DCW files are less useful because few of them contain complete information. Most people will have little need to read the raw DCW data files. If you need to do so, the Mapping Toolbox can read most DCW files into MATLAB structures using the `dcwread` function. The “External Data Reference” section of the *Mapping Toolbox Reference Guide* has more information on directly accessing individual DCW files.

More information on the relationships between the files and the meaning of the data in the files can be found in the “Military Specifications and Standards.”

U.S. Vector Data

One of the best sources of public vector data for the United States is the U.S. Census Bureau. This organization has developed a digital database and system for automated mapping in support of the United States decennial census. The Topographically Integrated Geographic Encoding and Referencing (TIGER) system is used within the Bureau to carry out and analyze the census. TIGER contains hydrographic features, roads, railroads, and other transportation features scanned from 1:100,000 U.S. Geological Survey (USGS) topographic maps. The database also contains census tracts and political boundaries for states, counties, and indian reservations; it does not include topography. The Census Bureau periodically publishes extracts from the TIGER database for public use in the form of TIGER/Line and TIGER thinned boundary files. The Mapping Toolbox provides interfaces to both formats.

TIGER/Line

TIGER/Line files are extracts from the U.S. Census Bureau's TIGER map database. Each set of files covers one municipal district and contains transportation features such as roads and railroads, cultural features, hydrographic features, and political boundaries. The data is distributed on CD-ROMs by the Census Bureau. TIGER/Line data is not copyrighted, but the Bureau does charge for the production of the CDs. Ordering and other information can be found from the U.S. Census Bureau's Web site at [<http://www.census.gov/>], along with sample TIGER/Line files. CD-ROMs may also be accessible over the Internet through various organizations.

You can import TIGER/Line data into MATLAB using the `tgrline` function. Files can be several thousand lines long, and reading them may require considerable time and memory. Once the data has been read and processed, you may want to save the results into a MATLAB workspace MAT-file for future use.

The following example reads the Washington, D.C., data from the 1994 TIGER/Line edition. These files are available over the Internet from The MathWorks at [ftp://ftp.mathworks.com/] and are delivered as text files in a compressed PC Zip archive. When extracting, make sure the text files are formatted for your computer platform. Generally, the names of the TIGER/Line datasets begin with the string 'tgr' and contain the 5-digit Federal Information Processing Standards (FIPS) code for the state and county.

The dataset for the District of Columbia is 'TGR1101':

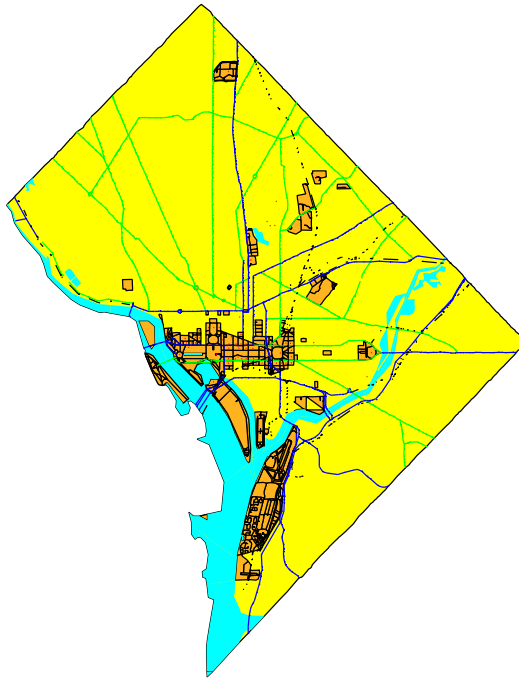
```
[CL, PR, SR, RR, H, AL, PL] = tgrline('TGR11001');
whos
```

Name	Size	Bytes	Class
AL	45x1	112952	struct array
CL	1x1	7444	struct array
H	16x1	25030	struct array
PL	64x1	59730	struct array
PR	1x32	53534	struct array
RR	1x8	16632	struct array
SR	1x40	76870	struct array

The `tgrline` function returns Mapping Toolbox geographic data structures for the county line (CL), primary roads (PR), secondary roads (SR), railroads (RR), hydrography (H), area landmarks (AL), and point landmarks (PL). This is a subset of the total available data in the TIGER/Line files, which contain more detailed local roads and other geographic reference data, such as Zip codes and census tract numbers, that are not handled by the function.

Since the data is now formatted into geographic data structures, you can display the objects using `di spl aym`. A Mercator projection is appropriate for small-scale, regional maps:

```
axesm mercator
di spl aym(CL)
hh = di spl aym(H); set(hh, 'EdgeColor', 'None')
di spl aym(AL); di spl aym(PL)
di spl aym(PR); di spl aym(SR); di spl aym(RR)
```



You can also produce the same map with graphical user interfaces. Using the `maptool` application, invoke the `mlayers` tool by selecting

Session⇒Layers⇒Workspace from the menu bar, or invoke `mlayers` from the command line by typing `rootlayr; mlayers(ans)`.

The amount of detail in the data becomes apparent when you zoom into the federal district of Washington, D.C. While the map has much detail, the street patterns contain displaced points, and the number of points in the paths of the streets is not enough to give smooth representations of slowly curving streets.



If you hold down the mouse button on an object, you will see its name appear in the lower left part of the figure window. All objects have been tagged with a name. Here are the names of places in the hydrography layer:

```
unique(strvcat(H. tag), ' rows')
```

```
ans =  
Anacostia River  
Channel  
Dalecarlia Reservoir  
Georgetown Reservoir  
Kenilworth Aquatic Gardens  
Kingman Lake  
Lagoon  
Mc Millan Reservoir  
Potomac River  
Reflecting Pool  
Tidal Basin  
Washington Channel
```

The other layers have tags with street names, interstate highway numbers, railroad names, and landmarks names. The names can be useful in finding the location of a site or in the selective display of data. What is the latitude and longitude of the Washington Monument? Mark the spot:

```
[lat, long, indx] = extractm(PL, 'washington mon');
PL(indx)
ans =

    lat: 38.8891
    long: -77.0354
    tag: 'Washington Monument'
    type: 'line'
    altitude: 0.4000
    otherproperty: {1x3 cell}

h = displaym(PL(indx)); set(h, 'Marker', 'x', 'Color', 'r')
```

You can also use the `objects` tool to browse the names of objects that have already been plotted.

TIGER Thinned Boundary

The U.S. Census Bureau has published lower resolution, thinned extracts of the TIGER database in two data file formats: ARC/INFO and MapInfo Interchange Format (MIF). ARC/INFO and MapInfo are geographic information systems (GIS) developed by the Environmental Systems Research Institute, Inc. (ESRI) and MapInfo Corporation, respectively. Data files for both formats are provided by the U.S. Census Bureau via the Internet [<http://www.census.gov/>], along with further descriptions and documentation.

TIGER ArcInfo Format

The following boundary types are available in the ArcInfo format: state, county, minor civil division, census tract/block numbering area, American Indian reservation/Alaska native village statistical area, Alaska native regional corporation, urbanized area, metropolitan area, and congressional district.

The ArcInfo data files have names ending in 'p' and 'pa'; the first represents polygon coordinate data, and the latter polygon attributes. You will also need the '_name.dat' file, or "names" file, containing the FIPS codes and corresponding county names.

Plot the counties in the state of Alaska as an example. You will need the following three files: 'co_name.dat,' 'co_02_p.dat,' and 'co_02_pa.dat.' For these files, 'co' indicates county boundaries, and '02' is the code for Alaska. The code '99' is used for files within the continental United States, and '15' corresponds to Hawaii. The files are compressed in a PC Zip format. When extracting, make sure the text files are formatted for your computer platform.

Read all U.S. county names and FIPS code from the names file. There are more than 3000 entries, so reading the file may take a while:

```
namestruc = fipsname(' co_name. dat' )
namestruc =
1x3248 struct array with fields:
    name
    id
```

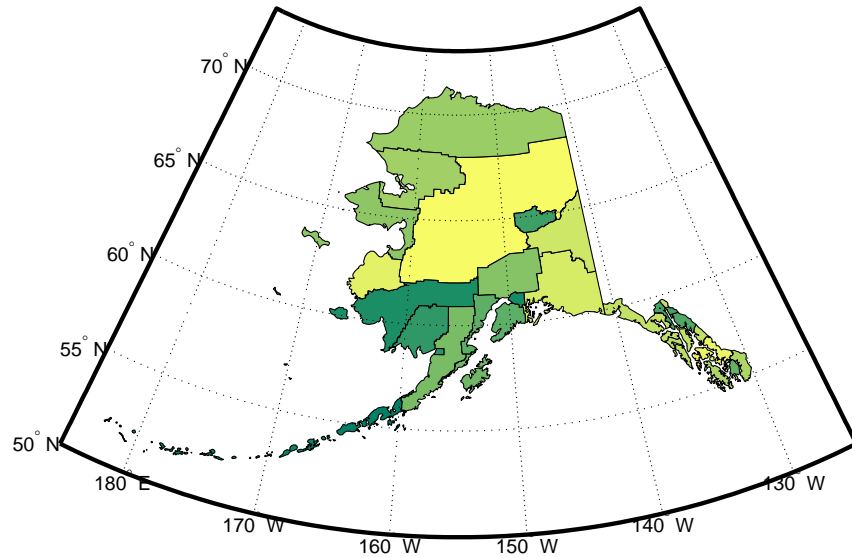
You can query the FIPS code of a particular city or county in Alaska, Anchorage, for example, by the following commands:

```
indx = strmatch(' Anchorage' , {namestruc. name})
indx =
    70
code = namestruc(indx).id
code =
    2020
```

Use the `tigerp` function to read and process Alaska's thinned county boundaries data into Mapping Toolbox geographic data structures.

Plot the data in a Lambert Conformal Conic projection:

```
[C0patch, C0text] = tigerp(namestruc, 'co_02_p.dat');
axesm('MapProjection', 'lambert', ...
      'MapLatLimit', [50 73], 'MapLonLimit', [175 235] , ...
      'MLabelLocation', 10, 'MLineLocation', 10, ...
      'PLabelLocation', 5, 'PLineLocation', 5, ...
      'MLabelParallel', 'south', 'MapParallels', []);
frame; gridm; mlabel; plabel
display(C0patch)
colormap(summer)
```



There is one entry in the C0patch structure for each county, with the county name in the tag field. The C0text structure contains the text labels for the counties.

You can also append new data to existing structures or extract only the counties you want. The following example reads the county boundaries for Nome and then adds the boundaries for the Aleutians East and Aleutians West to the output structure. You can look in the `namestruc` structure to find the codes you need.

```
C0patch = tigerp(namestruc, 'co_02_p.dat', [], [], 2180);
C0patch = tigerp(namestruc, 'co_02_p.dat', C0patch, [], ...
    [2013 2016])
C0patch =
1x3 struct array with fields:
    lat
    long
    type
    otherproperty
    altitude
    tag
```

TIGER MapInfo Interchange Format

The U.S. Census Bureau provides thinned boundary files in the MapInfo Interchange Format. Both U.S. state and county boundaries are available in this format.

The MapInfo data files have the 'mif' and 'mid' extensions. You will also need the '_name.dat' file, or "names" file, containing the Federal Information Processing Standard (FIPS) code numbers and corresponding county names.

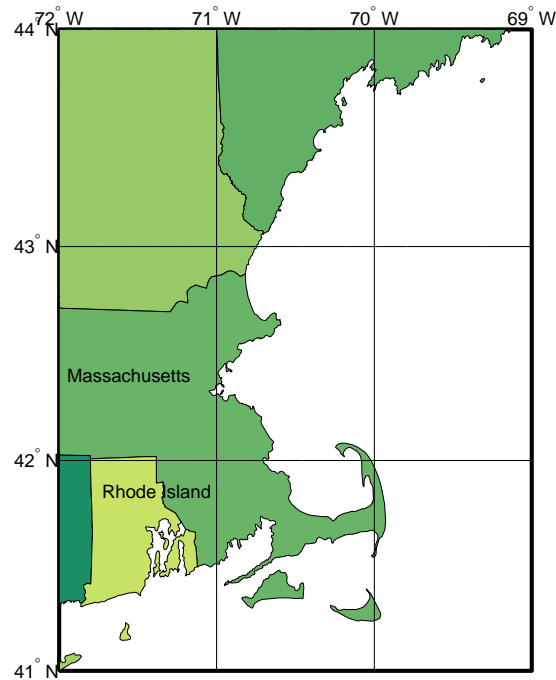
The following example reads the state boundaries of Alaska and Hawaii. Five files are needed: 'ST02.MID,' 'ST02.MIF,' 'ST15.MID,' 'ST15.MIF,' and 'st_name.dat.' You must first read the names file for the names and FIPS codes for the U.S. states and territories:

```
namestruc = fipsname('st_name.dat')
namestruc =
1x57 struct array with fields:
    name
    id
```

Read the file containing Hawaii's county boundaries into a geographic data structure, and append it with Alaska's state boundaries:

```
STpatch = tigermif(namestruc, 'ST15.MIF');
STpatch = tigermif(namestruc, 'ST02.MIF', STpatch)
STpatch =
1x2 struct array with fields:
    lat
    long
    type
    otherproperty
    tag
    altitude
```

Finally, to compare the resolution and quality of this data to other sources, look at the region around Cape Cod. This map may look familiar to you; it is the same data in the `usahi` workspace, plotted in a Mercator projection. The `usahi` data was derived from the Census Bureau's TIGER Thinned Boundary files.



Global Gridded Elevation Data

Digital Elevation Models (DEMs) represent another class of useful geographic data. These are matrices of land elevations and ocean depths, generally on a regular grid. The `topo` workspace in MATLAB is an example of a low resolution DEM, with cells that extend 1 degree by 1 degree, giving a spatial resolution of about 100 kilometers or better. The Mapping Toolbox provides interfaces to more detailed DEMs, ranging in resolution from 10 kilometers to 100 meters.

ETOPO5 and TerrainBase

ETOPO5 and TerrainBase are digital elevation models with worldwide coverage at a resolution of 5 minutes (10 kilometers or better). They are both compilations of data from a variety of different sources, including the U.S. Naval Oceanographic Office, U.S. Defense Mapping Agency, U.S. Navy Fleet Numerical Oceanographic Center, Bureau of Mineral Resources in Australia, and the Department of Industrial and Scientific Research in New Zealand. The ETOPO5 dataset was assembled by Margo Edwards at Washington University, in St. Louis, Missouri.

An overview of this dataset can be found at several Internet locations, including the U.S. Geological Survey Web site located at [<http://edcwww.cr.usgs.gov/>] and the U.S. National Geophysical Data Center home page at [<http://www.ngdc.noaa.gov/>].

The ETOPO5 data is read using the `etopo5` external interface function for the version of the database contained in the two text files, 'etopo5.southern.bat' and 'etopo5.northern.bat.' These files occupy about 60 megabytes of total disk space when uncompressed and are provided via FTP by the U.S. Geological Survey at [<ftp://ftp:walrus.wr.usgs.gov/>] and by The MathWorks, Inc. at [<ftp://ftp.mathworks.com/>].

ETOPO5 has been superseded by TerrainBase, which is similar in coverage to ETOPO5, but corrects many of its flaws, including discontinuities at joins between datasets, systematic shifts in location, and lack of data in some regions. TerrainBase is a compilation of the best available, public domain data from almost 20 different sources, including ETOPO5 and DCW-DEM, another digital elevation model that has a Mapping Toolbox interface, covered in the next section.

The TerrainBase dataset was created by the National Geophysical Data Center and World Data Center-A for Solid Earth Geophysics in Boulder, Colorado. The model is currently under development and will be updated as new data sources become available. Further information on the TerrainBase DEM can be found at the U.S. National Geophysical Data Center home page at [<http://www.ngdc.noaa.gov/>].

TerrainBase data is read using the `tbase` function. The data files are available on CD-ROM and via Internet. NOAA/NGDC provides ordering information and downloadable files at their Web site. The files require about 18 megabytes of disk space when uncompressed.

You should use the TerrainBase dataset unless you have a good reason to want ETOPO5. TerrainBase corrects some errors, takes up less space, and can be read faster than ETOPO5. The following examples will read TerrainBase data, but you can substitute ETOPO5 data by simply using the `etopo5` function instead of `tbase`.

A 10 kilometer resolution DEM has so much data that it is usually impractical to read the entire file into memory. The full matrix has 2160 by 4320 elements. Both the `etopo5` and `tbase` functions allow you to reduce the resolution by discarding data points.

Read every twelfth point in the global data base.

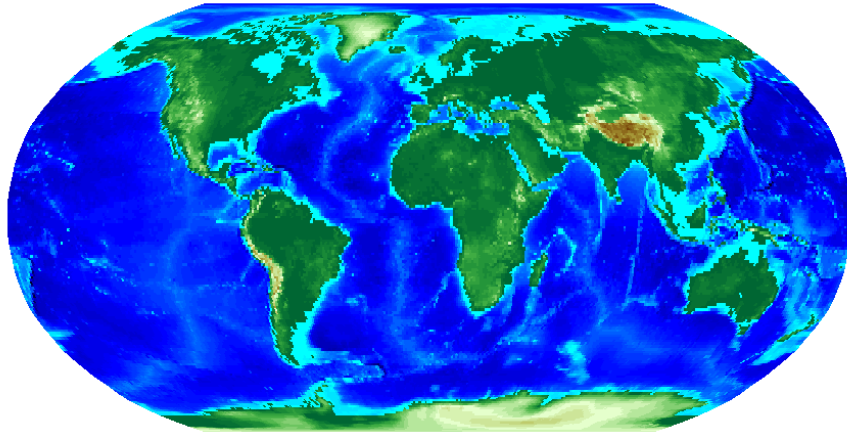
```
[map, maplegend] = tbase(12);
```

```
whos
```

Name	Size	Bytes	Class
map	180x360	518400	double array
maplegend	1x3	24	double array

Plot the map on a Robinson projection:

```
axesm robinson  
meshm(map, maplegend); demcmap(map);
```



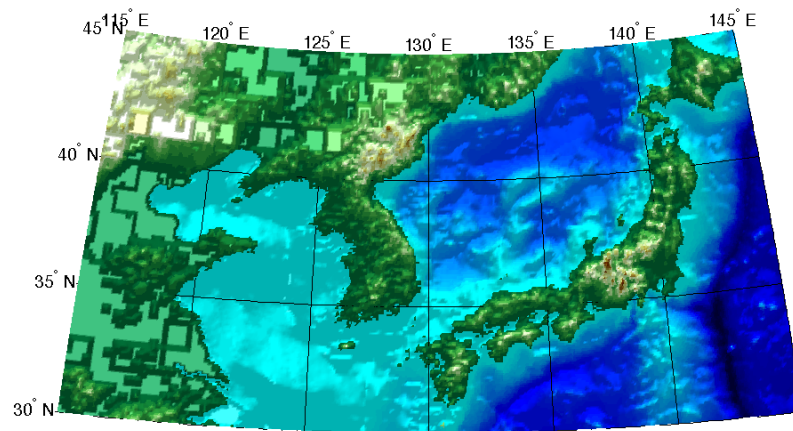
This is a regular matrix map at the same resolution as the topo map. It differs from topo in that the values represent nearest neighbor samples, rather than averages over the matrix cell, resulting in a rougher appearance.

Elevations and depths are given in meters above or below mean sea level. Some parts of the world are represented by data with a horizontal resolution as coarse as 1 degree by 1 degree. The vertical resolution varies from 1 meter for Australia and New Zealand to as much as 150 meters for parts of Africa, Asia, and South America. Oceanographic data in areas shallower than 200 meters contains little detail because of the way depth contours were converted to gridded depths.

The areas with 1 degree horizontal data can be clearly seen when you display data for a small region at the full resolution of the database.

Display the Korean peninsula and Japan:

```
[map, maplegend] = tbase(1, [30 45], [115 145]);
axesm('MapProjection', 'polycon', ...
      'MapLatLimit', [30 45], 'MapLonLimit', [115 145], ...
      'MLabelLocation', 5, 'MLineLocation', 5, ...
      'PLabelLocation', 5, 'PLineLocation', 5, ...
      'GLineStyle', '-', 'GAltitude', inf) ;
gridm; mlabel; plabel
meshm(map, maplegend, size(map), map); demcmap(map);
lightm(37.5, 130); material([.7 .7 1.5]); lighting gouraud
```



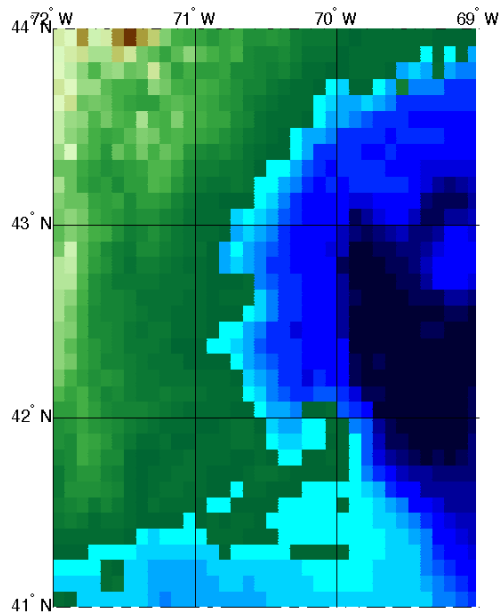
Note the discontinuities in some areas, particularly the regions in China.

You can get a sense for the resolution of this dataset by comparing a TerrainBase map of the Cape Cod region in the Northeastern United States with maps of the same region using other datasets. TerrainBase and ETOPO5 have a horizontal resolution of 12 cells per degree, so the Cape Cod map consists of 36 by 36 cells.

```
[map, maplegend] = tbase(1, [41 44], [-72 -69]);
whos

      Name                Size      Bytes    Class
      map                  36x36      10368    double array
      maplegend             1x3         24      double array

axesm('Mapprojection', 'mercator', ...
      'MapLatLimit', [41 44], 'MapLonLimit', [-72 -69], ...
      'MLabelLocation', 1, 'MLineLocation', 1, ...
      'PLabelLocation', 1, 'PLineLocation', 1, ...
      'GLineStyle', '-', 'GAltitude', inf) ;
gridm; mlabel; plabel
meshm(map, maplegend); demcmap(map)
```



Digital Chart of the World DEMs

Another source of gridded elevation data is the collection of DEMs derived from the Digital Chart of the World (DCW), commonly referred to as the DCW-DEM. The Mapping Toolbox provides an interface to both the DCW and the DEMs derived from it. See the section entitled “Global Vector Data” for more information on the DCW.

The DCW-DEM is currently under development by the Earth Resources Observation Systems (EROS) Data Center. The elevations are computed for a regular grid using the ANUDEM computer program from the Australian National University. This program combines elevation and drainage lines and points to interpolate elevations at the grid locations.

The dataset provides global coverage of land elevations at a horizontal resolution of 30 arc-second (about 1 kilometer or better). The DCW is based on aeronautical charts; hence, no bathymetric data is available.

The DCW-DEM is accessible over the Internet and can be purchased on CD-ROM as major geographic regions are completed. Ordering information, as well as the actual data files, can be found at the EROS Data Center's home page at [<http://edcwww.cr.usgs.gov/>]. Preliminary versions of Africa, Antarctica, Asia, Europe, North America, and South America are currently available. Samples of Japan, Madagascar, and Haiti may also be available.

The Mapping Toolbox interface to DCW-DEM files is the `dcwdem` function. It reads the elevations into regular matrix maps, optionally downsampling and trimming data in the process.

The following examples make use of the Haiti sample files, 'haiti.hdr' and 'haiti.bil,' which are available from the EROS Data Center's FTP server at [ftp://edcftp.cr.usgs.gov/] or The MathWorks at [ftp://ftp.mathworks.com/].

Read the entire Haiti dataset at one-fourth resolution. Large negative values in the map represent a lack of data rather than actual depth and have been changed to an appropriate value.

Elevations are given in feet above mean sea level using WGS 84 as a horizontal datum.

```
[map, maplegend] = dcwdem('haiti', 4);
```

```
map(map<0) = -1;
```

```
whos
```

Name	Size	Bytes	Class
map	121x331	320408	double array
maplegend	1x3	24	double array

```
axesm mercator
```

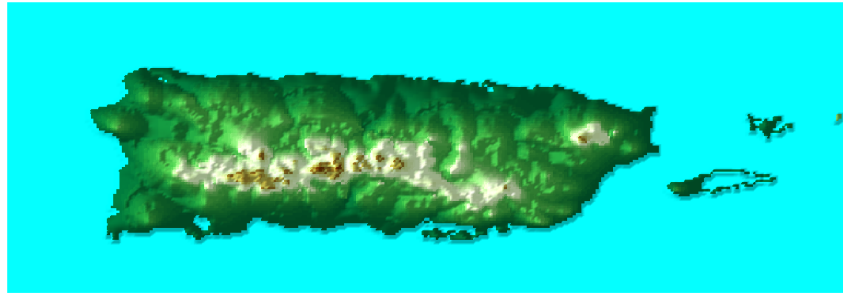
```
meshm(map, maplegend, size(map), map); demcmap(map)
```

```
lightm(19, -69.5); material([.7 .7 1.5]); lighting gouraud
```



You can read a subset of the data by providing latitude and longitude limits. Here is the part of the Haiti dataset covering Puerto Rico at the full resolution (30 arc-seconds).

```
[map, maplegend] = dcwdem('hai ti', 1, ...  
                           [17.75 18.75], [-67.5 -65.05]);  
map(map<0) = -1;  
  
axesm mercator  
meshm(map, maplegend, size(map), map); demcmap(map)  
lightm(18.25, -66.275); material([.7 .7 1.5])  
lighting gouraud
```

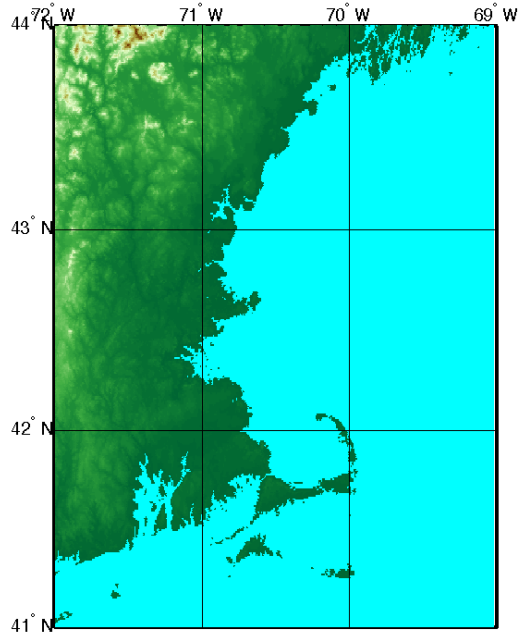


The increase in resolution over the ETOPO5 and TerrainBase datasets can be appreciated from a map of the Cape Cod region. The 30 arc-second DCW-DEM data covers this region with a 360-by-360 matrix, which is 10 times finer than the 5 arc-minute (300 arc-second) ETOPO5 and TerrainBase data. This is also the same resolution as the data in the MATLAB `cape` workspace.

```
[map, maplegend] = dcwdem('na_30_dem3', 1, [41 44], [-72 -69]);
map(isnan(map)) = -1;
whos
```

Name	Size	Bytes	Class
map	360x360	1036800	double array
maplegend	1x3	24	double array

```
axesm('MapProjection', 'mercator', ...
      'MapLatLimit', [41 44], 'MapLonLimit', [-72 -69], ...
      'framem', 'gridm('MLineLocation', 1, 'PlineLocation', 1)
      'mlabel('MLabelLocation', 1); plabel('PLabelLocation', 1)
      meshm(map, maplegend); demcmap(map)
```



U.S. Gridded Elevation Data

The highest resolution digital elevation maps accessible through the Mapping Toolbox external interface functions cover most of the United States at a horizontal resolution of about 100 meters. They are distributed by the U.S. Geological Survey as separate 1-degree quadrangles covering the contiguous United States, Hawaii, and limited portions of Alaska. The 1-degree DEMs are also referred to as “3-arc second” or “1:250,000 scale” DEM data.

The data is derived from the U.S. Defense Mapping Agency's DTED-1 digital elevation model, which itself was derived from cartographic and photographic sources. The cartographic sources are maps from the 7.5 minute through 1-degree series (1:24,000 scale through 1:250,000 scale). There is no bathymetric data in this DEM.

Most of the 1-degree digital elevation models have grid spacings of 3 arc-seconds in both the latitude and longitude directions. Alaska has grid spacings of 6 arc-seconds for latitudes between 50 and 70 degrees North and 9 arc-seconds for latitudes greater than 70 degrees North.

The grid for the digital elevation maps is based on the World Geodetic System 1984 (WGS 84). Older DEMs were based on WGS 72. Elevations are in meters relative to the National Geodetic Vertical Datum of 1929 (NGVD 29) in the continental U.S. and local mean sea level in Hawaii.

The specified absolute horizontal accuracy of the DEM is 130 m, while the specified absolute vertical accuracy is ± 30 m. The relative horizontal and vertical accuracies are not specified but should be much better than the absolute accuracies.

The DEM data files can be retrieved over the Internet from the EROS Data Center's Web site located at [<http://edcwww.cr.usgs.gov/>] or their FTP server at [<ftp://edcftp.cr.usgs.gov/>]. The files are available sorted by state and file name or can be selected from an index map. Further information on the dataset is also available at the sites listed.

The data files are named according to towns or features within the quadrangle region. If you know the geographic limits of the area you want to map, you can look up the names on the USGS Web site or use the Mapping Toolbox `usgsdems` function, which returns a list of map quadrangle filenames covered by your region. Here are the names of the files covered by the Cape Cod maps used throughout this document:

```
usgsdems([41 44], [-72 -69])
ans =
    'providence-w'
    'providence-e'
    'chatham-w'
    'boston-w'
    'boston-e'
    'portland-w'
    'portland-e'
    'bath-w'
```

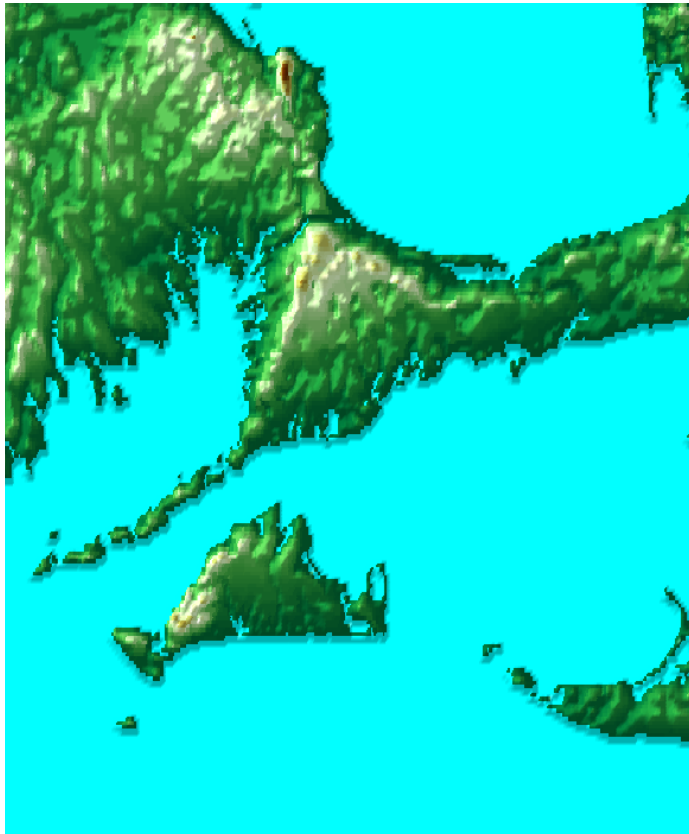
The Cape Cod region extends over several of the 1-degree quadrangles. Displaying the whole region at the full, 3 arc-second resolution would require enormous amounts of memory and time.

Display the region covered by the 'providence-e' quadrangle using the `usgsdem` interface function and a downsampling factor of 5. The resulting map is a 241-by-241 element matrix. At full resolution, the map would be 1201-by-1201.

```
[map, maplegend] = usgsdem('providence-e', 5);
map(map==0) = -1;

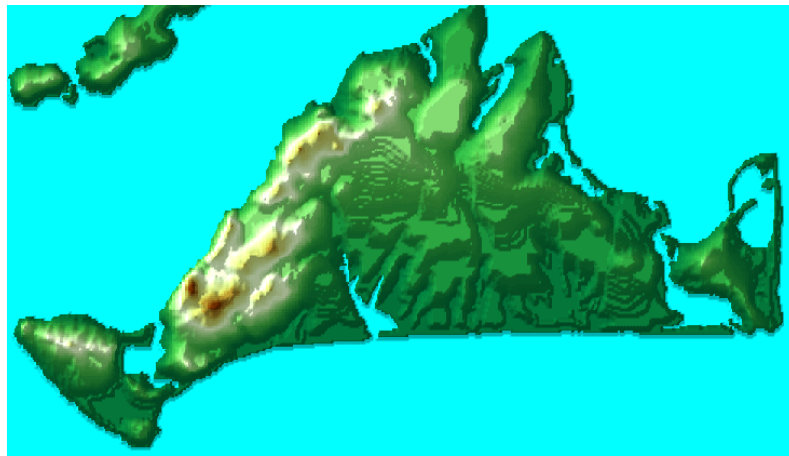
axesm mercator
meshm(map, maplegend, size(map), map); demcmap(map)
lightm(41.5, -70.5); material([.7 .7 1.5]); lighting gouraud
```

This is a zoomed-in view of the immediate region around Cape Cod Bay and Martha's Vineyard:



Zoom in further on Martha's Vineyards by providing appropriate latitude and longitude limits for the `usgsdem` function at full resolution. You can see some terracing effects in the data, which correspond to its derivation from topographic contour maps.

```
[map, maplegend] = usgsdem(' providence-e' , 1, [41.2946 41.4829], ...  
                           [-70.8429 -70.4379]);  
axesm mercator  
meshm(map, maplegend, size(map), map); demcmap(map)  
lightm(41.4, -70.65); material([.7 .7 1.5]); lighting gouraud
```



Astronomical Data

The Mapping Toolbox provides an interface to a set of astronomical data called “FK5”, the Fifth Fundamental Catalog of Stars. This catalog consists of positions, proper motions, and other characteristics for over 4500 stars. The data is provided by the Astronomical Data Center (ADC), located at the NASA Goddard Space Flight Center, and can be retrieved over the Internet at the ADC’s home page at [<http://adc.gsfc.nasa.gov/>] or by anonymous FTP at [<ftp://adc.gsfc.nasa.gov/>]. Documentation and other information can be found at these sites as well.

The data comes in two files, ‘FK5.dat’ and ‘fk5_ext.dat.’ You can read both files using the `readfk5` interface function. There are more than four thousand stars in the catalog, so reading and processing the data takes a while.

```
fk5 = readfk5('FK5.dat');
fk5e = readfk5('fk5_ext.dat');
whos
```

Name	Size	Bytes	Class
fk5	1x1535	5042752	struct array
fk5e	1x3117	10226424	struct array

The star catalog is processed into MATLAB structures by the external interface function. The fields of the structures contain the different types of information for each star, while the structure elements represent the individual stars.

To view the fields of the structure, type the following:

```
fk5(1)
ans =
    FK5: 1
    RAh: 0
    RAm: 8
    RAs: 23. 2650
    pmRA: 1. 0390
    DEd: 29
    DEm: 5
    DEs: 25. 5800
    pmDE: - 16. 3300
    RAh1950: 0
    RAm1950: 5
    RAs1950: 47. 8770
    pmRA1950: 1. 0360
    DEd1950: 28
    DEm1950: 48
    DEs1950: 51. 9600
    pmDE1950: - 16. 3300
    EpRA1900: 43. 3100
    e_RAs: 0. 7000
    e_pmRA: 2
    EpDE1900: 33
    e_DEs: 1. 3000
    e_pmDE: 3. 1000
    Vmag: 2. 0600
    n_Vmag: ''
    SpType: 'A0p'
    pl x: 0. 0240
    RV: - 11. 7000
    AGK3R: ''
    SRS: ''
    HD: ' 358'
    DM: ' BD+28      4'
    GC: ' 127'
```

If you just want to display the locations of the stars, you can find position and visual magnitude data in the `stars` workspace as part of the Mapping Toolbox atlas data. The remainder of this section illustrates the use of Mapping Toolbox functions to manipulate and transform star data for display.

The star positions are given in terms of right ascension and declination. You can display a star map by converting the positions to latitude and longitude coordinates. Right ascension is given in hours, minutes, and seconds, while declination is in degrees, minutes, and seconds.

```
hRA = [fk5. RAh fk5e. RAh]'; dDE = [fk5. DEd fk5e. DEd]';  
mRA = [fk5. RAm fk5e. RAm]'; mDE = [fk5. DEm fk5e. DEm]';  
sRA = [fk5. RAs fk5e. RAs]'; sDE = [fk5. DEs fk5e. DEs]';  
lat = dms2deg(dDE, mDE, sDE);  
lon = hms2hr(hRA, mRA, sRA) * 360 / 24;
```

You probably want the map to look like you are inside the celestial sphere looking out, so switch the sign of the longitudes:

```
lon = zero22pi(-lon);
```

It is customary to display the stars with sizes proportional to the visual magnitudes. Extract the magnitudes from the data structures:

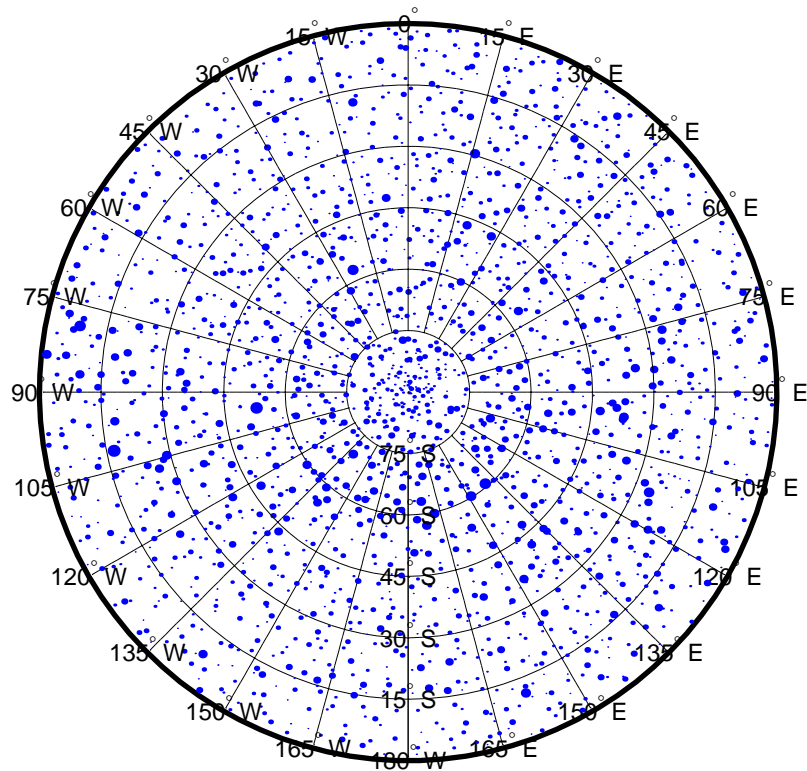
```
magn = [fk5. Vmag fk5e. Vmag]';
```

Faint stars have large visual magnitude numbers, while bright ones have smaller values. The magnitudes of particular bright stars were used to define the visual magnitude scale. It was later determined that certain stars were even brighter than the reference stars, resulting in negative visual magnitudes. You will need to convert the values to positive numbers for plotting.

```
pvmag = 0.01 + max(magn) - magn;  
clear fk5 fk5e
```

Stars in equatorial coordinates are typically displayed in the Equidistant Azimuthal projection. Here is the southern sky. You can see the southern cross at about 60° S, 175° E.

```
axesm eqdazim
framem; gridm; mlabel; plabel
setm(gca, 'Origin', [-90 180 0], 'FLatLimit', [-inf 90], ...
      'MLineLocation', 15, 'MLabelLocation', [-180 15: 165], ...
      'MLabelParallel', 'equator', 'MLineLimit', [-75 75], ...
      'PLabelLocation', [-15: -15: -75], ...
      'GLineWidth', .01, 'GLineStyle', '-')
scatterm(lat, lon, pvmag, 'b', 'MarkerSize', .0125)
```



The sky is often depicted in galactic coordinates, with the center of our galaxy as the origin of a polar coordinate system. You can use the Mapping Toolbox to transform the star positions from equatorial coordinates to galactic coordinates. First define the location of the galactic pole and the galactic center.

```
% galactic pole
pol eRAhms = mat2hms([12 49 0]);
pol eDEhms = mat2hms([27 24 0]);
pol el at = dms2deg(pol eDEhms);
pol el on = zero22pi (-hms2hr(pol eRAhms)*360/24);

% galactic center
cntrRAhms = mat2hms([17 42 0.4]);
cntrDEhms = mat2hms([-28 55 0]);
cntrl at = dms2deg(cntrDEhms);
cntrl on = zero22pi (-hms2hr(cntrRAhms)*360/24);
```

Next create a new origin vector based on the location of the new pole in the old equatorial coordinate system. Use it to rotate the locations of the pole and the center of the galactic coordinates. Note that the origin provided by the `newpole` function results in a system centered on the location of the new pole in the old coordinate system. Rotate the star positions and bring the galactic center to zero longitude.

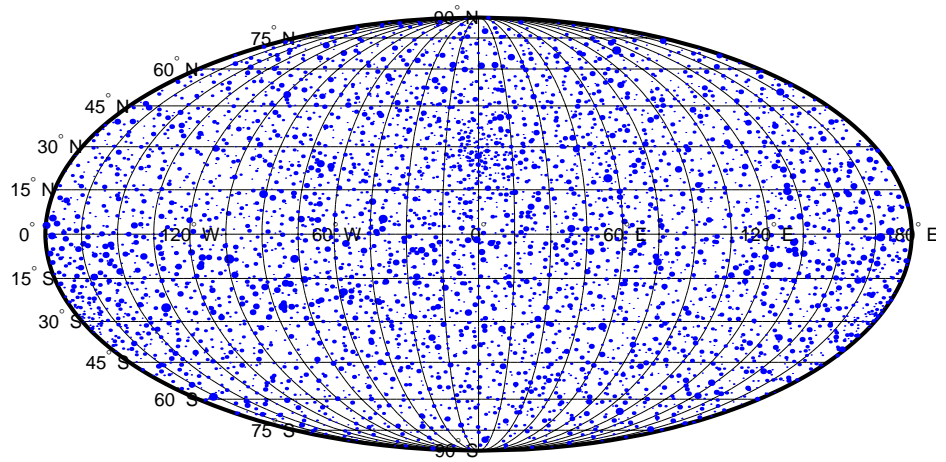
```
% new origin vector
ori g i n = newpole(pol el at, pol el on);

% rotate our known points
[newpol el at, newpol el on] = rotatem(pol el at, pol el on, ...
                                     ori g i n, 'forward', 'degrees');
[newcntrl at, newcntrl on] = rotatem(cntrl at, cntrl on, ...
                                     ori g i n, 'forward', 'degrees');

% rotate the stars
[gl at, gl on] = rotatem(l at, l on, ori g i n, 'forward', 'degrees');
gl on = gl on - newcntrl on;
```


The stars in galactic coordinates are commonly displayed in a pseudo-cylindrical projection like the Mollweide.

```
figure
axesm mollweid
frame; gridm; mlabel; plabel
setm(gca, 'MLabelLocation', -120:60:180,...
        'MLineLocation', 15, 'MLabelParallel', 'equator',...
        'GLineWidth', .01, 'GLineStyle', '-',...
        'FLatLimit', [-inf 90])
scatterm(gl at, gl on, pvmag, 'b', 'MarkerSize', .0125)
```



Geographic Terms

Glossary Notes

This glossary of geographical terms is drawn extensively from *An Album of Map Projection*, *U.S. Geological Survey Professional Paper 1453*, by John P. Snyder and Philip M. Voxland.

The purpose of this glossary is to assist the user in understanding the Mapping Toolbox. For this reason, some terms in this glossary are specifically defined as they are used in this guide rather than as they might be more generally used in the geographic community.

Glossary

Antipodes - Two points on opposite sides of a planet.

Arc-second - 1/3600th of a degree (1 second) of latitude or longitude.

ARC/Info - A largely UNIX-oriented GIS developed and distributed by the Environmental Systems Research Institute, Inc.

Aspect - The conceptual placement of a projection system in relation to the Earth's axis (direct, normal, polar, equatorial, oblique, and so on).

Authalic projection - *See* Equal-area projection.

Axes - *See* Map axes.

Azimuth - The angle a line makes with a meridian, taken clockwise from north.

Azimuthal projection - A projection on which the azimuth or direction from a given central point to any other point is shown correctly. When a pole is the central point, all meridians are spaced at their true angles and are straight radii of concentric circles that represent the parallels. Also called a zenithal projection.

Bathymetry - The measurement of water depths of oceans, seas, lakes, and other bodies of water.

Bowditch, Nathaniel - A late 18th/early 19th century mathematician, astronomer, and sailor who "wrote the book" on navigation. John Hamilton Moore's *The Practical Navigator* was the leading navigational text when Bowditch first went out to sea, and had been for many years. Early in his first voyage, however, Bowditch began noticing errors in Moore's book, which he recorded and later used in preparing an American edition of Moore's work. The revisions were to such an extent that Bowditch was named the principal author, and the title was changed to *The New American Practical Navigator*, published in 1802. In 1868 the U.S. Navy bought the copyright to the book, which is still commonly referred to as "Bowditch" and considered the "bible" of navigation.

Cartography - The art or practice of making charts or maps.

Central meridian - The meridian passing through the center of a projection, often a straight line about which the projection is symmetrical.

Central projection - A projection in which the Earth is projected geometrically from the center of the Earth onto a plane or other surface. The Gnomonic and Central Cylindrical projections are examples.

Choropleth - A map consisting of areas of equal value separated by abrupt boundaries and colored or shaded according to those values.

Complex curves - Curves that are not elementary forms such as circles, ellipses, hyperbolas, parabolas, and sine curves.

Composite projection - A projection formed by connecting two or more projections along common lines such as parallels of latitude, necessary adjustments being made to achieve fit. The Goode Homolosine projection is an example.

Conformal projection - A projection on which all angles at each point are preserved. Also called an orthomorphic projection.

Conceptually projected - The convenient way to visualize a projection system, although it may not correspond to the actual mathematical projection method.

Conic projection - A projection resulting from the conceptual projection of the Earth onto a tangent or secant cone, which is then cut lengthwise and laid flat. When the axis of the cone coincides with the polar axis of the Earth, all meridians are straight equidistant radii of concentric circular arcs representing the parallels, but the meridians are spaced at less than their true angles. Mathematically, the projection is often only partially geometric.

Constant scale - A linear scale that remains the same along a particular line on a map, although that scale may not be the same as the stated or nominal scale of the map.

Contour - All points that are at the same elevation above or below a specified datum.

Conventional aspect - *See* Normal aspect.

Correct scale - A linear scale having exactly the same value as the stated or nominal scale of the map, or a scale factor of 1.0. Also called true scale.

Cylindrical projection - A projection resulting from the conceptual projection of the Earth onto a tangent or secant cylinder, which is then cut lengthwise and laid flat. When the axis of the cylinder coincides with the axis of the Earth, the meridians are straight, parallel, and equidistant, while the parallels of latitude are straight, parallel, and perpendicular to the meridians. Mathematically, the projection is often only partially geometric.

Dead reckoning - From “deduced reckoning,” the estimation of geographic position based on course, speed and time.

DEM (Digital Elevation Map/Model) - Elevation data in the form of a matrix map, generally on a regular grid. DEM also refers to the five primary types of digital elevation models produced by the U.S. Geological Survey, one of which is the 1-degree (3-arc-second resolution) model that is interfaced through the Mapping Toolbox.

Departure - The arc length distance along a parallel of a point from a given meridian.

Developable surface - A simple geometric form capable of being flattened without stretching. Many map projections can be grouped by a particular developable surface: cylinder, cone, or plane.

Direct aspect - *See* Normal aspect.

Distortion - A variation of the area or linear scale on a map from that indicated by the stated map scale, or the variation of a shape or angle on a map from the corresponding shape or angle on the Earth.

DMS - Degrees-minutes-seconds angle notation of the form ddd° mm' ss". There are 60 seconds in a minute, and 60 minutes in a degree. In the Mapping Toolbox, when “dms” angles are represented by a single number, the format is dddmm.ss.

Ellipsoid - When used to represent the Earth, a solid geometric figure formed by rotating an ellipse about its minor (shorter) axis. Also called spheroid.

Equal-area projection - A projection on which the areas of all regions are shown in the same proportion to their true areas. Shapes may be greatly distorted. Also called an equivalent or authalic projection.

Equatorial aspect - An aspect of an azimuthal projection on which the center of projection or origin is some point along the Equator. For cylindrical and pseudocylindrical projections, this aspect is usually called conventional, direct, normal, or regular rather than equatorial.

Equidistant projection - A projection that maintains constant scale along all great circles from one or two points. When the projection is centered on a pole, the parallels are spaced in proportion to their true distances along each meridian.

Equireal projection - *See* Equal-area projection.

Equivalent projection - *See* Equal-area projection.

Flat-polar projection - A projection on which, in normal aspect, the pole is shown as a line rather than as a point.

Frame - *See* Map frame.

Free of distortion - Having no distortion of shape, area, or linear scale. On a flat map, this condition can exist only at certain points or along certain lines.

General matrix map - In the Mapping Toolbox, a matrix map defined with latitude-longitude coordinate matrices, allowing irregular, non-rectangular orientations.

Geoid - The true shape of the Earth, an irregular and complex shape which is usually modeled with a sphere or an ellipsoid. In the Mapping Toolbox, this term refers to the spherical or ellipsoidal model of the Earth *or other planet* in use rather than the true shape.

Geoid vector - In the Mapping Toolbox, a vector describing the geoid, or ellipsoid, model. The geoid vector has the form:

`geoidvec = [semi major-axis eccentricity]`

Geometric projection - *See* Perspective projection.

Geographic data structure - In the Mapping Toolbox, A MATLAB data structure containing the data and other information for the proper display of map objects. Valid fields in the structure include type, tag, and altitude.

GIS (Geographic Information System) - A system, usually computer based, for the input, storage, retrieval, analysis, and display of interpreted geographic data.

Globular projection - Generally, a nonazimuthal projection developed before 1700 on which a hemisphere is enclosed in a circle and meridians and parallels are simple curves or straight lines.

Graticule - A network of lines representing a selection of the Earth's parallels and meridians for the purpose of projection. The vertices of the graticule grid are precisely projected, and the map data contained in any grid cell is warped to fit the resulting quadrilateral. A finer graticule grid results in a higher projection fidelity at the expense of greater computational requirements.

Great circle - Any circle on the surface of a sphere, especially when the sphere represents the Earth, formed by the intersection of the surface with a plane passing through the center of the sphere. It is the shortest path between any two points along the circle and therefore important for navigation. All meridians and the Equator are great circles on the Earth taken as a sphere.

Grid - *See* Map grid.

HMS - Hours-minutes-seconds time notation of the form hh° mm' ss". In the Mapping Toolbox, when "hms" times are represented by a single number, the format is hhmm.ss.

Homolographic/homolographic projection - *See* Equal-area projection.

Hydrography - The science of measurement, description, and mapping of the surface waters of the Earth, especially with reference to their use in navigation. The term also refers to those parts of a map collectively that represent surface waters.

Hydrology - The scientific study of the waters of the Earth, especially with relation to the effects of precipitation and evaporation upon the occurrence and character of ground water.

Hypsography - The scientific study of the Earth's topologic configuration above sea level, especially the measurement and mapping of land elevation.

Indexed map - A matrix map in which entries are an index value into another data source. The world map workspace contains an example of an indexed map. Each entry in the matrix map is an index into a data structure containing the names of the world countries.

Indicatrix - A circle or ellipse having the same shape as that of an infinitesimally small circle (having differential dimensions) on the Earth when it is plotted with finite dimensions on a map projection. Its axes lie in the directions of and are proportional to the maximum and minimum scales at that point. This is useful in illustrating the distortions of a given map projection. Often called a Tissot indicatrix after the originator of the concept. In the Mapping Toolbox, Tissot indicatrices may be displayed using the `tissot` command, and indicatrices for all supported projections are provided in the “Projections Reference” section of the *Mapping Toolbox Reference Guide*.

Interrupted projection - A projection designed to reduce peripheral distortion by making use of separate sections joined at certain points or along certain lines, usually the Equator in the normal aspect, and split along lines that are usually meridians. There is normally a central meridian for each section. The Mapping Toolbox does not include interrupted projections, but the user can separate data into sections and project these independently to achieve this effect.

Large-scale mapping - Mapping at a scale larger than about 1:75,000, although this limit is somewhat flexible.

Latitude (geographic) - The angle made by a perpendicular to a given point on the surface of a sphere or ellipsoid representing the Earth and the plane of the Equator (positive if the point is north of Equator, negative if it is south). One of the two common geographic coordinates of a point on the Earth.

Latitude of opposite sign - *See* Parallel of opposite sign.

Legs - Line segments connecting waypoints.

Legend - *See* Map legend.

Limiting forms - The form taken by a system of projection when the parameters of the formulas defining that projection are allowed to reach limits that cause it to be identical with another separately defined projection.

Logical map - A binary matrix map consisting entirely of 1s and 0s. An example of a logical matrix map can be created with the `topo` map by performing a logical test for positive elevations (`topo>0`). Each entry in the matrix map contains a 1 if it is above sea level, or a 0 if it is at or below sea level.

Longitude - The angle made by the plane of a meridian passing through a given point on the Earth's surface and the plane of the (prime) meridian passing through Greenwich, England, east or west to 180 (positive if the point is east, negative if it is west). One of the two common geographic coordinates of a point on the Earth.

Loxodrome - *See* Rhumb line.

Map A representation of geographic data. In the Mapping Toolbox, a map is any variable or set of variables (electronically) representing or assigning values to a geographic location or region, from a single point to an entire planet.

Map axes - In the Mapping Toolbox, a normal MATLAB axes altered for mapping display purposes. Several map axes properties are defined and stored in the UserData slot of the MATLAB axes. These properties control the appearance of the map display, much like the properties of the normal MATLAB axes control the appearance of the displayed plot. A map axes must first be defined in order to display maps using the Mapping Toolbox.

Map frame - In the Mapping Toolbox, a projected "box" or quadrangle enclosing the geographic display.

Map grid - A displayed network of lines representing parallels and meridians. The grid is used for visual reference and should not be confused with the graticule.

Map legend - In the Mapping Toolbox, a vector defining the geographic placement and unit cell size of a regular matrix map. A map legend has the form:

```
maplegend = [cells/angle unit north-latitude west-longitude]
```

MapInfo - A largely PC-oriented GIS developed and distributed by the MapInfo Corporation.

Matrix map - A map consisting of a matrix (or matrices) of values corresponding to specific geographic points. In the Mapping Toolbox, matrix maps can be defined as regular or general, depending on the structure and orientation of the geographic points. *See* Regular matrix map and General matrix map.

Meridian - A reference line on the Earth's surface formed by the intersection of the surface with a plane passing through both poles and some third point on the surface. This line is identified by its longitude. On the Earth as a sphere, this line is half a great circle; on the Earth as an ellipsoid, it is half an ellipse.

Minimum-error projection - A projection having the least possible total error of any projection in the designated classification, according to a given mathematical criterion. Usually, this criterion calls for the minimum sum of squares of deviations of linear scale from true scale throughout the map ("least squares").

NGVD 29 (National Geodetic Vertical Datum of 1929) - A reference surface established by the U.S. Coast Guard and Geodetic Survey of 1929, used as the datum for which relief features and elevation data are referenced in the conterminous United States; formerly called "mean sea level 1929."

Nominal scale - The stated scale at which a map projection is constructed.

Normal aspect - A form of a projection that provides the simplest graticule and calculations. It is the polar aspect for azimuthal projections, the aspect having a straight Equator for cylindrical and pseudocylindrical projections, and the aspect showing straight meridians for conic projections. Also called conventional, direct, or regular aspect.

Oblique aspect - An aspect of a projection on which the axis of the Earth is rotated so it is neither aligned with nor perpendicular to the conceptual axis of the map projection.

Orthoapsidal projection - A projection on which the surface of the Earth taken as a sphere is transformed onto a solid other than the sphere and then projected orthographically and obliquely onto a plane for the map.

Orthographic projection - A specific azimuthal projection or a type of projection in which the Earth is projected geometrically onto a surface by means of parallel projection lines.

Orthomorphic projection - *See* Conformal projection.

Parallel - A small circle on the surface of the Earth formed by the intersection of the surface of the reference sphere or ellipsoid with a plane parallel to the plane of the Equator. This line is identified by its latitude. The Equator (a great circle) is usually also treated as a parallel.

Parallel of opposite sign - A parallel that is equally distant from but on the opposite side of the Equator. For example, for lat 30°N (or +30°), the parallel of opposite sign is lat 30° S (or -30°). Also called latitude of opposite sign.

Parameters - The values of constants as applied to a map projection for a specific map; examples are the values of the scale, the latitudes of the standard parallels, and the central meridian. The required parameters vary with the projection.

Perspective projection - A projection produced by projecting straight lines radiating from a selected point (or from infinity) through points on the surface of a sphere or ellipsoid and then onto a tangent or secant plane. Other perspective maps are projected onto a tangent or secant cylinder or cone by using straight lines passing through a single axis of the sphere or ellipsoid. Also called geometric projection.

Planar projection - A projection resulting from the conceptual projection of the Earth onto a tangent or secant plane. Usually, a planar projection is the same as an azimuthal projection. Mathematically, the projection is often only partially geometric.

Planimetric map - A map representing only the horizontal positions of features (without their elevations).

Polar aspect - An aspect of a projection, especially an azimuthal one, on which the Earth is viewed from the polar axis. For cylindrical or pseudocylindrical projections, this aspect is called transverse.

Pole - An extremity of a planet's axis of rotation. The North Pole is a singular point at 90°N for which longitude is ambiguous. The South Pole has the same characteristics and is located at 90°S.

Polyconic projectio - A specific projection or member of a class of projections that are constructed like conic projections but with different cones for each parallel. In the normal aspect, all the parallels of latitude are nonconcentric circular arcs, except for a straight Equator, and the centers of these circles lie along a central axis.

Projection - A systematic representation of a curved 3-D surface such as the Earth onto a flat 2-D plane. Each map projection has specific properties that make it useful for specific purposes.

Pseudoconic projection - A projection that, in the normal aspect, has concentric circular arcs for parallels and on which the meridians are equally spaced along the parallels, like those on a conic projection, but on which meridians are curved.

Pseudocylindrical projection - A projection that, in the normal aspect, has straight parallel lines for parallels and on which the meridians are (usually) equally spaced along parallels, as they are on a cylindrical projection, but on which the meridians are curved.

Quadrangle - A region bounded by parallels north and south, and meridians east and west.

Raster map - *See* Matrix map.

Reckoning - The determination of geographic position by calculation.

Regional map - A small-scale map of an area covering at least 5 or 10 degrees of latitude and longitude but less than a hemisphere.

Regular aspect - *See* Normal aspect.

Regular matrix map - In the Mapping Toolbox, an equiangular (equal-angle) matrix map defined with a map legend vector, limited to a rectangular orientation.

Retroazimuthal projection - A projection on which the direction or azimuth from every point on the map to a given central point is shown correctly with respect to a vertical line parallel to the central meridian. The reverse of an azimuthal projection.

Rhumb line - A complex curve (a spherical helix) on a planet's surface that crosses every meridian at the same oblique angle; a navigator can proceed between any two points along a rhumb line by maintaining a constant heading. A rhumb line is a straight line on the Mercator projection. Also called a loxodrome.

Scale - The ratio of the distance on a map or globe to the corresponding distance on the Earth; usually stated in the form 1:5,000,000 for example.

Scale factor - The ratio of the scale at a particular location and direction on a map to the stated scale of the map. At a standard parallel, or other standard line, the scale factor is 1.0.

Secant cone, cylinder, or plane - A secant cone or cylinder intersects the sphere or ellipsoid along two separate lines; these lines are parallels of latitude if the axes of the geometric figures coincide. A secant plane intersects the sphere or ellipsoid along a line that is a parallel of latitude if the plane is at right angles to the axis.

Shaded Relief - Shading added to a map or image that makes it appear to have three-dimensional aspects. This type of enhancement is commonly done to satellite images and thematic maps utilizing digital topographic data to provide the appearance of terrain relief.

Similar (projection) - Subjective and qualitative term indicating a moderate or strong resemblance.

Singular points - Certain points on most but not all conformal projections at which conformality fails, such as the poles on the normal aspect of the Mercator projection.

Skew-oblique aspect - An aspect of a projection on which the axis of the Earth is rotated, so it is neither aligned with nor perpendicular to the conceptual axis of the map projection, and tilted, so the poles are at an angle to the conceptual axis of the map projection.

Small circle - A circle on the surface of a sphere formed by the intersection with a plane. Parallels of latitude are small circles on the Earth taken as a sphere. In the Mapping Toolbox, great circles, including the Equator and all meridians, are treated as special, limiting cases of small circles.

Small-scale mapping - Mapping at a scale smaller than about 1:1,000,000, although the limiting scale sometimes has been made as large as 1:250,000.

Spheroid - *See* Ellipsoid.

Standard parallel - In the normal aspect of a projection, a parallel of latitude along which the scale is as stated for that map. There are one or two standard parallels on most cylindrical and conic map projections and one on many polar stereographic projections.

Stereographic projection - A specific azimuthal projection or type of projection in which the Earth is projected geometrically onto a surface from a fixed (or moving) point on the opposite face of the Earth.

Tangent cone or cylinder - A cone or cylinder that just touches the sphere or ellipsoid along a single line. This line is a parallel of latitude if the axes of the geometric figures coincide.

Thematic map - A map designed to portray primarily a particular subject, such as population, railroads, or croplands.

Tissot indicatrix - *See* Indicatrix.

Topographic map - A map that usually represents the vertical positions or elevations of features as well as their horizontal positions. The topo workspace contains a simple example.

Transformed latitudes, longitudes, or poles - Graticule of meridians and parallels on a projection after the Earth has been turned with respect to the projection so that the Earth's axis no longer coincides with the conceptual axis of the projection. Used for oblique and transverse aspects of many projections.

Transverse aspect - An aspect of a map projection on which the axis of the Earth is rotated so that it is at right angles to the conceptual axis of the map projection. For azimuthal projections, this aspect is usually called equatorial rather than transverse.

True scale - *See* Correct scale.

Valued map - A matrix map in which entries represent some value or measurement. The topo workspace contains an example of a valued map. Each entry in the matrix map is an average elevation in meters for the geographic position represented by that cell.

Vector map - A map consisting of ordered latitude-longitude points, possibly connected. In the Mapping Toolbox, such map data is often represented by two vectors, representing latitude and longitude. Segments can be separated by the insertion of NaN's in both vectors.

Waypoint - Points through which a track passes, usually corresponding to course or speed changes.

WGS 72 (World Geodetic System 1972) - An Earth-centered datum, used as a definition of DMA DEMs, presently stored in the USGS data base. The WGS 72 datum was the result of an extensive effort extending over approximately three years to collect selected satellite, surface gravity, and astrogeodetic data available throughout 1972. These data were combined using a unified WGS solution (a large-scale least squares adjustment).

WGS 84 (World Geodetic System 1984) - The WGS 84 was developed as a replacement for the WGS 72 by the military mapping community as a result of new and more accurate instrumentation and a more comprehensive control network of ground stations. The newly developed satellite radar altimeter was used to deduce geoid heights from oceanic regions between 70° north and south latitude. Geoid heights were also deduced from ground-based Doppler and ground-based laser satellite-tracking data, as well as surface gravity data.

Zenithal projection - *See* Azimuthal projection.

Bibliography

- 1 Snyder, J. P., *Map Projections - A Working Manual*, U.S. Geological Survey Professional Paper 1395, Washington, D.C., 1987.
- 2 Maling, D. H., *Coordinate Systems and Map Projections*, 2nd Edition, Pergamon Press, New York, NY, 1992.
- 3 Snyder, J. P. and Voxland, P. M., *An Album of Map Projections*, U.S. Geological Survey Professional Paper 1453, Washington, D.C., 1994.
- 4 Snyder, J. P., *Flattening the Earth - 2000 Years of Map Projections*, University of Chicago Press, Chicago, IL, 1993.

A

- almanac 1-19
- almanac data 1-19
- angle units
 - convention for navigation functions 5-12
 - converting between formats 1-22
 - description of formats 1-21
- angl edim 1-23
- anti pole 1-9
- areaint 1-32
- areamat 1-51
- areaquad 1-15
- aspect *See* projection aspect
- astronomical data 3-30
- axes *See* map axes
- axes2ecc 1-17
- axesm 2-3, 2-11, 2-15
- azimuth
 - defined 1-11
 - measuring 1-11
- azi muth 1-11
- azimuthal projection 4-6

B

- base projection 4-13
- bearing *See* azimuth

C

- cape workspace 3-28
- central projection 4-6
- changem 1-48
- choropleth maps, example 2-52
- circles
 - See* great circles
 - See* small circles

- Click-and-drag editor 6-15
- coast workspace 1-3, 3-3
- col orm 6-41
- colormap
 - creating for digital elevation map 2-34
- colormaps
 - creating with GUI 6-41
 - menu in mapprool 6-4
- cometm 2-50
- composite maps 1-6
- conformal projection 4-3
- conic projection 4-5
- coordinate transformations 4-14
 - example using stars 7-39
 - matrix data 4-16
 - vector data 4-15
- cylindrical projection 4-4

D

- dcwdata 7-5
- dcwdem 7-28
- DCW-DEM data 7-28
- dcwgaz 7-4
- dead reckoning 5-31
 - calculating positions 5-33
 - rules of 5-32
- deg2km 1-11
- deg2nm 1-24
- deg2rad 1-8
- degrees notation 1-21
- demcmap 2-34
- departure 5-5
- Digital Chart of the World 7-3
 - extracting data 7-5
 - looking up names 7-3

- reading data files directly 7-13
- digital elevation maps (DEMs)
 - coloring 2-34
 - defined 1-4
- distance 1-10
- distance *See* surface distance
- distance units
 - convention for navigation functions 5-12
 - converting between formats 1-24
 - description of formats 1-24
- dstdim 1-24
- dms notation 1-21
- dms2deg 1-22
- dms2rad 1-22
- dreckon 5-12, 5-33

E

- Earth
 - default geoid 1-18
 - ellipsoid models 1-18
- ellipsoid
 - as a geoid model 1-16
 - converting parameters 1-17
 - models for Earth 1-18
 - models for planets 1-19
- eqa2grn 5-10
- equal-area projection 4-3
- equidistant projection 4-3
- etopo5 7-23
- ETOPO5 model 7-23
- external data sources 7-2
- extractm 2-46

F

- Federal Information Processing Standard (FIPS)
 - 7-19, 7-21
- Federal Information Processing System (FIPS)
 - 7-15
- Fifth Fundamental Catalog of Stars 7-36
- fillm 2-28, 2-29
- filtering geographic data 5-11
- filterm 5-11
- findm 1-48
- fipsname 7-19, 7-21
- fixing *See* navigational fixing
- frame *See* map frame
- framem 2-11, 2-14

G

- gcwaypts 5-12, 5-28
- gcxgc 1-34
- gcxsc 1-34
- general matrix maps
 - defined 1-53
 - displaying 2-30
 - image and surface coloring 2-36
 - light shading 2-37
 - shaded relief 2-39
 - See also* matrix maps
- geographic data structure
 - defined 2-43
 - displaying 2-45
 - extracting data from 3-11, 3-21
- geographic mean 5-3
- geographic standard deviation 5-5
- geoid
 - availability for planets 1-19
 - converting ellipsoid parameters 1-17
 - defined 1-16

- ellipsoid approximation 1-16
- ellipsoid models for Earth 1-18
- geoid vector 1-17
- geostatistics
 - calculating geographic mean 5-3
 - calculating geographic standard deviation 5-5
 - equal-area coordinate system 5-10
 - equirectangular binning 5-8
 - filtering data sets 5-11
 - histograms 5-8
- getm 2-6
- gnomonic projection 4-6
- graphical user interfaces
 - activating 6-2
 - getting help 6-2
 - See also* maptool
- graticule
 - choosing resolution 2-32, 2-33
 - defined 1-56, 2-31
- great circles
 - approximating tracks with rhumb lines 5-28
 - calculating points of 1-28
 - defined 1-9
- grid *See* map grid
- grn2eqa 5-10

H

- handl em 2-49
- hi dem 2-49
- histograms 5-8
- hi str 5-8
- hms notation 1-23
- hours notation 1-23

I

- indexed maps
 - defined 1-50
 - modifying colormap 6-41
 - replacing values 6-33
- inputm 2-47, 5-29, 6-26
- interplat 1-31
- interplon 1-31
- interp m 1-30
- interpolation, latitude and longitude 1-30

K

- korea workspace 2-36

L

- labels
 - meridians and parallels 2-17
 - text 2-22
- latitude and longitude
 - interpolation 1-30
 - See also* map frame, setting limits
 - See also* map limits
- latitude, defined 1-7
- legs
 - course and distance of 5-30
 - defined 5-13
- legs 5-12, 5-30
- light objects, displaying 2-40
- lightm 2-40
- limit m 1-44
- line objects
 - displaying 2-24
 - plotting from maptool 6-13, 6-24
- loadcape 3-28
- logical maps

- calculating surface area 1-51
 - defined 1-51
- longitude
 - converting between ranges 1-8
- longitude, defined 1-8
- loxodromes *See* rhumb lines
- l t l n 2 v a l 1-47

M

- map axes
 - accessing properties 2-6
 - current values 2-7
 - default values 2-9
 - defined 2-3
 - defining a map projection 2-3
 - differences from standard axes 2-5
 - resetting to default properties 2-21
 - setting properties 2-3, 2-6
- map definition 1-2
- map frame
 - adjusting for a new projection 2-19
 - controlling appearance 2-14
 - defined 2-11
 - displaying 2-11
 - resetting altitude 2-14
 - setting limits 2-12, 2-18, 6-22
 - using maptool 6-30
- map grid
 - controlling appearance 2-15
 - defined 2-15
 - displaying 2-15
 - resetting altitude 2-15
 - setting properties 6-6, 6-22
 - using maptool 6-30
- map layers
 - plotting 6-7
- map legend
 - defined 1-43
- map limits
 - adjusting for a new projection 2-19
 - setting 2-13, 2-18
 - versus frame limits 2-18
- map projection
 - aspect 4-7
 - azimuthal 4-6
 - base 4-13
 - central 4-6
 - choosing 4-22
 - conformal 4-3
 - conic 4-5
 - cylindrical 4-4
 - defined 2-3, 4-2
 - defining for a map axes 2-3
 - distortion 4-3
 - equal-area 4-3
 - equidistant 4-3
 - gnomonic 4-6
 - orthographic 4-6
 - polyconic 4-5
 - stereographic 4-6
 - switching 2-23
 - table of properties 4-22
- mapmt x workspace 1-53
- mapped object
 - versus standard object 2-22
- mapped objects
 - changing stacking order 6-28
 - editing properties 6-9
 - manipulating by name 2-48
- maps 2-3
- maptool 6-2
 - activating 6-3, 6-5
 - calculating surface distance 6-16

- default map projection 6-4
- displaying map grid and frame 6-30
- entering workspace commands 6-12
- hiding mouse tool buttons 6-29
- hiding objects 6-20
- menus 6-4
- mouse tool buttons 6-4, 6-29
- plotting line objects 6-13, 6-24
- plotting patches 6-27
- plotting text objects 6-14, 6-24
- plotting tracks 6-16
- setting map projection 6-5, 6-22
- setting map properties 6-5, 6-22
- zoom feature 6-11
- maptrim 6-31
- maptriml 1-36
- maptrimp 1-36
- maskm 1-48
- mat2dms 1-21
- mat2hms 1-23
- matrix maps
 - coloring 2-34, 6-41
 - defined 1-4
 - displaying 2-30
 - graticules 2-31
 - indexed maps 1-50, 6-33, 6-41
 - logical maps 1-51
 - replacing entries 1-48, 6-33
 - See also* general matrix maps
 - See also* regular matrix maps
 - valued maps 1-49
- mean of geographic data 5-3
- meanm 5-4
- Mercator projection 4-13, 5-13, 5-28
- meridians
 - controlling display 2-15
 - defined 1-8

- meshgrat 1-59, 2-36
- meshl srm 2-39, 2-42
- meshm 6-42
- minaxis 1-18
- mlayers 2-48, 6-7, 6-27
- mobiles 6-9
- mouse
 - interaction with displayed maps 2-47
 - selecting points with 6-26

N

- namem 2-49
- nanm 1-51
- navfix 5-12, 5-19
- navigation
 - angular conventions 5-12
 - calculating course and distance 5-30
 - calculating dead reckoning positions 5-33
 - calculating waypoints 5-28
 - connecting waypoints 5-29
 - distance conventions 5-12
 - fixing position 5-13, 5-19
 - retrieving time zone for longitude 5-36, 5-38
 - speed conventions 5-12
- navigational fixing 5-13, 5-19
- navigational tracks
 - connecting waypoints 5-29
 - displaying 5-28
 - displaying from maptool 6-27
 - format 5-13
 - plotting from maptool 6-16
 - selecting waypoints with mouse 6-26
- neworig 4-16
- newpole 4-15, 4-16
- normal aspect 4-7
- npi2pi 1-8

O

oblique aspect 4-8
onem 1-51
orientation 4-10
orientation *See* origin vector
origin vector 4-7, 4-10
 See also projection aspect
orthographic projection 4-6

P

parallels
 controlling display 2-15
 defined 1-7
patch drawing functions
 differences between 2-29
patch objects
 displaying 2-27
 plotting from maptool 6-27
patchesm 2-29
patchm 2-29
piloting *See* navigational fixing
planetary data 1-19
plot3m 2-28
plotm 2-24
polyconic projection 4-5
polygon
 displaying as line object 2-24
 displaying as patch object 2-27
 surface area 1-32
projection *See* map projection
projection aspect 4-7
 normal 4-7
 oblique 4-8
 See also origin vector
 skew-oblique 4-12
 transverse 4-8

projection control dialog box
 setting map projection 6-5, 6-22
 setting map properties 6-5, 6-22

Q

qui verm 2-50

R

radians notation 1-21
radius of planets 1-20
raster maps *See* matrix maps
readfk5 7-36
reckon 1-13
reckoning position 1-13
reducem 1-38
reducing data *See* vector data
regular matrix maps
 accessing elements 1-47
 defined 1-42
 determining limits 1-44
 determining size with scaling 1-49
 displaying 2-30, 6-42
 image and surface coloring 2-36
 shaded relief 2-39
 encoding 6-33
 See also matrix maps
rhumb lines
 approximating great circle tracks with 5-28
 calculating points 1-28
 defined 1-9
rhxrh 1-34
rotatem 4-15
russia workspace 1-44

S

- scatterm 2-50, 5-8
- scircle1 1-27
- scircle2 1-27
- scxsc 1-34
- seconds notation 1-23
- seedm 6-33
- setltn 1-45
- setm 2-6, 2-11, 2-15
- setpostn 1-45
- shaded relief maps 2-39
- shown 2-49
- simplifying data *See* vector data
- size 1-49
- skew-oblique aspect 4-12
- small circles
 - calculating points 1-27
 - defined 1-13
- speed units
 - format for navigation functions 5-12
- spzerm 1-51
- stacking order of objects, changing 6-28
- standard deviation of geographic data 5-5
- standard parallels
 - for conic projections 2-28
- stars workspace 3-30, 7-38
- stdist 5-6
- stdm 5-5
- stem plot, example 2-50
- stem3m 2-50
- stereographic projection 4-6
- surface area
 - accessing from almanac 1-20
 - measuring
 - logical maps 1-51
 - polygons 1-32
 - quadrangles 1-15

- surface distance

- calculating from maptool 6-16
 - measuring 1-10

- surface objects, displaying 2-30

- surfdist 6-18

- surflm 2-37

- surflrm 2-39, 2-42

- surfm 2-42

- symbol plot, example 2-51

T

- tbase 7-24

- TerrainBase model 7-23

- text objects

- editing with Click-and-Drag editor 6-15
 - plotting from maptool 6-14, 6-24

- tgrline 7-14

- TIGER Thinned data 7-18

- TIGER/Line data 7-14

- tigermf 7-22

- tigerp 7-19

- time units

- conventions for navigation 5-38
 - converting between formats 1-24
 - description of formats 1-23

- time zones

- navigational 5-36

- time2str 5-39

- timedim 1-24

- timezone 5-36, 5-38

- tissot 2-50

- topo workspace 1-5, 3-17

- topographical maps

- See* digital elevation maps (DEMs)

- track 5-29, 6-26

- track1 1-28

track2 1-28, 2-26
 tracks
 See great circles
 See rhumb lines
 trackui 6-16
 transformation of coordinate system
 See coordinate transformation
 transverse aspect 4-8
 trimming data 1-36, 6-31

U

U.S. matrix data
 political 3-26
 terrain 3-28
 U.S. vector data
 low resolution 3-18
 medium resolution 3-23
 units
 See angle units
 See distance units
 See time units
 usahi workspace 3-23
 usal o workspace 2-27, 2-44, 3-18
 usamt x workspace 3-26
 UserData property
 used by the Mapping Toolbox 2-2, 2-3
 USGS DEM data 7-32
 usgsdem 7-33
 usgsdems 7-33

V

valued maps
 defined 1-49
 vector data
 calculating intersections 1-34

 creating 1-25, 1-27
 geographic interpolation 1-30
 simplifying/reducing 1-38
 trimming data to a region 1-36
 vector maps
 data format of 1-25
 defined 1-3
 delineation of objects in 1-25
 displaying as lines 2-24
 displaying as patches 2-27
 volume of planets 1-20

W

waypoints
 calculating 5-28
 connecting 5-29
 defined 5-13
 selecting with mouse 5-29, 6-26
 world matrix data
 finding countries from coordinates 3-15
 political data 3-14
 terrain data 3-17
 world vector data
 atlas data 3-5
 coastline data 3-3
 deleting data with tags 3-9
 displaying atlas data 3-5
 extracting data with tags 3-11
 worl dl o workspace 3-5
 activating from maptool 6-7
 worl dmt x workspace 3-14

Z

zero22pi 1-8
 zerom 1-51

zoom GUI 6-11